arXiv:2509.18057v7 [cs.LG] 9 Mar 2026

Reinforced Generation of Combinatorial Structures: Hardness of Approximation

Ansh Nagda* Prabhakar Raghavan† Abhradeep Thakurta‡

Abstract

Can AI based methods help us make advances in complexity theory? We provide evidence towards answering this in the affirmative, using AlphaEvolve (an LLM code mutation agent) to obtain new results in three settings:

a) Average-case hardness for MAX-CUT and MAX-Independent Set: We improve a recent result of Kunisky and Yu to obtain near-optimal upper and (conditional) lower bounds on certification algorithms for MAX-CUT and MAX-Independent Set on random 3- and 4-regular graphs. Our improved lower bounds are obtained by constructing nearly extremal Ramanujan graphs on as many as 163 vertices. Additionally, via analytical arguments we strengthen the upper bounds to settle the computational hardness of these questions up to an error in the third decimal place.

b) Worst-case Hardness of Approximation for MAX-k-CUT: We obtain new inapproximability results, proving that it is NP-hard to approximate MAX-4-CUT and MAX-3-CUT within factors of 0.987 and 0.9649 respectively, using AlphaEvolve to discover new gadget reductions. Our MAX-4-CUT result improves upon the SOTA of 0.9883, and our MAX-3-CUT result improves on the current best gadget-based inapproximability result of 0.9853, but falls short of improving the SOTA of 16/17 that relies on a custom PCP (rather than a gadget reduction from "standard" Håstad-style PCPs).

c) Worst-case Hardness of Approximation for the metric Traveling Salesman Problem (TSP): We show that it is NP-hard to approximate the minimum cost tour within a factor of 111/110 using AlphaEvolve to discover a new gadget, thus improving the SOTA of 117/116. We provide a modular soundness and completeness argument for the reduction of 3LIN(2) (a standard constraint satisfaction problem used in hardness reductions) to TSP, which enabled AlphaEvolve to search over finite constraint graphs. This modularization may be of independent interest for future work on TSP inapproximability.

A key technical challenge we faced: verifying a candidate construction produced by AlphaEvolve is costly (sometimes requiring time exponential in the size of the construction). Our results were enabled by using AlphaEvolve itself to evolve the verification procedure to be faster (sometimes by $10,000\times$ for our gadgets). Our results suggest that gadget based proofs would benefit from a pass through AI-based tools to obtain stronger results.

*University of California, Berkeley, and Google DeepMind.
†Google.
‡Google DeepMind.

Contents

1 Introduction **1**

2 Hardness of Certification Problems on Random Graphs **2**

3 Gadget based NP-Hardness for Approximating MAX-k-CUT **5**
3.1 Developing a Search Framework for Soundness and Completeness Guarantees 8
3.2 Faster Verification via AlphaEvolve to Explore Larger Gadgets 9
3.3 Comparison to other computational techniques 10

4 NP-Hardness of Approximating Metric TSP **10**

5 Discussion on AI-assisted Mathematics and Complexity Theory **12**

6 Acknowledgments **15**

A AlphaEvolve as a Framework for Combinatorial Discovery **19**

B Proofs of Average Case Theorems **21**
B.1 Proof of Theorem 2.1 21
B.2 Proof of Theorem 2.2 22

C Proofs of Theorems 3.1 and 3.2 **25**
C.1 Description of our gadgets 27

D Proof of the Hardness of Approximation for TSP **29**
D.1 Proofs of Lemmas D.2 and D.3 32
D.2 Description of our gadget 36
D.3 Equation gadget for predicate $x + y + z \equiv 0 \pmod 2$ 36

1 Introduction

In this paper, we study the following question: Can AI based methods help us make novel and non-trivial discoveries in complexity theory? In particular, we seek to focus on classical and well-studied problems to derive results that stand the test of time. We are interested in proofs and constructions that will plausibly defy non-AI methods (whether by direct human effort, or through traditional computational methods such as SAT solvers). We provide evidence towards answering this in the affirmative, via progress on three problems.

a) *Hardness of certifying upper bounds on the* MAX-CUT *and* MAX-IND-SET *of sparse random graphs.* In Section 2, we improve upon the combinatorial structures presented in [KY24], achieving improved lower bounds in the $\{3,4\}$-regular cases. Complementing this result, we obtain new upper bounds which improve upon Hoffman's classic spectral bounds [Hof03, Hae21]. (These upper bounds are derived via analytical approaches[1].) Together, these achieve an almost tight resolution to these problems.[2].

b) *NP-Hardness of approximation for* MAX-k-CUT. In Section 3, we give new gadget-based reductions (from standard Håstad PCPs [Hås01]) for the NP-hardness of approximating MAX-k-CUT for $k \in \{3,4\}$. For MAX-4-CUT, we improve on the current best inapproximability result from 0.9883 [AOTW14] to 0.987. For MAX-3-CUT, we improve on the current best gadget-based inapproximability result from 0.9853 [KKLP96] to 0.9649. Our result for MAX-3-CUT is sandwiched between two results using custom PCPs for this problem: [GS09] with 0.9696, and [AOTW14] with 0.9411.

c) *NP-Hardness of approximation for metric TSP.* In Section 4, we give a new gadget based reduction (from standard Håstad PCPs [Hås01]) for the NP-hardness of approximating metric TSP. We improve the current SOTA of 117/116 [CC20] to 111/110. Our improvement advances the long line of work establishing hardness results for this problem [PY93, $1+\delta$] (without an explicit value for δ), [Eng99, 5381/5380], [BHK$^+$00, 3183/3182], [PV06, 220/219], [Lam14, 185/184], [KLS15, 123/122], [CC20, 117/116].

Notably, our results rely on a single (AI-derived) technique, AlphaEvolve [NVE$^+$25, RPBN$^+$24], to find and verify finite combinatorial structures that improve on prior constructions. At a high level AlphaEvolve uses an LLM (Large-Language-Model) to iteratively evolve code snippets that generate combinatorial structures (we will sometimes refer to these as *constructions*) that are of high quality by some criterion. Typically this process uses a synthetic objective function that is a *proxy* for the true objective in the theorem being sought; for instance for MAX-k-CUT and the TSP we seek to improve the constant in the hardness theorems. AlphaEvolve cannot directly optimize this constant, because of the manner in which proofs embed gadgets; rather, AlphaEvolve improves gadgets in a "good direction" that we specify and then — given a generated gadget — we reconstruct the resulting hardness guarantee. These mechanics will become evident in Sections 3 and 4. While the structures we generate using AlphaEvolve are finite constructions, our main results (Theorems 2.1, 3.1, and 4.1) embed the universal quantifier $\forall n$, with n being the size of the problem instance, thanks to appropriate "lifting" [KY24, TSSW00, CC20] arguments. At a minimum our work suggests that we would do well to routinely run gadget proofs through an "AI optimization" phase. We present our results in increasing order of human involvement (required to make the problem amenable to AlphaEvolve).

Operationally, the system consists of three parts: a) a code snippet ($\mathcal{C}$) that constructs combinatorial structures, b) An evaluation function (referred to as the *verifier*) that verifies and scores the generated structures, and c) An LLM that suggests a new code snippet $\mathcal{C}_{\texttt{new}}$ based on the set of prior $\mathcal{C}$'s and the history of constructions. Via prompting the LLM, the objective is to nudge $\mathcal{C}_{\texttt{new}}$ towards building better structures. (Appendix A provides a more detailed overview of AlphaEvolve.)

The efficacy of AlphaEvolve crucially depends on the verification step which in turn conditions the search landscape. Because we seek extremal structures in a large search space, *fast* verification (including scoring) of a construction helps AlphaEvolve try a large number of combinatorial structures, and learn patterns from them to prune the search space. The eventual sizes of the structures we find for our problems

[1] These improvements are derived from a discussion with Sidhanth Mohanty, Northwestern University.

[2] By this we mean: the lower bounds and upper bounds on efficient certification algorithms match up to the second decimal place.

are directly correlated with the speed of verification.

Speeding up verification in AlphaEvolve: The combinatorial structures sought in these problems are based on undirected graphs. While brute-force verification of our structures was reasonably fast for the TSP lower bound and for certifying upper bounds on sparse random graphs, it was significantly more challenging for MAX-k-CUT. We discuss this in more detail below.

For TSP, one of the discovered gadgets had 20 edges (over 12 vertices) along with a large number of constraints ($\approx$ 11!) for a successful verification. As a result, we needed some care in writing a speedy verifier that operated within roughly one second.

For the hardness of certifying upper bounds for sparse random graphs, we use AlphaEvolve to search for $d = \{3, 4\}$-regular Ramanujan graphs, along with a witness for a large cut/independent set. Evaluating whether a graph is Ramanujan, and whether or not the cut/independent set is a valid witness, are efficient. Consequently, for these problems we could find and verify graphs with as many as 163 vertices.

In studying the NP-Hardness of approximation for MAX-k-CUT, we seek gadgets that reduce the 3LIN(k) problem [Hås01] to MAX-k-CUT (3LIN(k) corresponds to a set of constraints of the form $x + y + z \equiv b \pmod{k}$, where $x, y, z \in [0, \ldots, k-1]$). Here the verification is unfortunately not efficient. It requires checking the gadget against $\Omega(k^m)$ linear constraints, where m is the total number of variables in the target problem. A standard technique in gadget-based hardness reductions is to add auxiliary variables to have flexibility in mapping the source and the target problems [TSSW00]. This exacerbates the exponential running time of the verifier (for reference, our final gadgets contain $m = 14$ and $m = 19$ variables for $k = 3$ and $k = 4$ respectively). To address this we invoke AlphaEvolve to speed up the verifier, as follows.

We prompted AlphaEvolve to improve the execution time of the verifier, while ensuring that it passes various correctness checks. To our surprise, AlphaEvolve improved the running time by $10,000\times$ for our final MAX-4-CUT gadget of size $m = 19$, and by a similar factor for our final MAX-3-CUT gadget of size $m = 14$. We used human inspection to verify the correctness of these improved verifiers (given their possibility of being correct only on synthetically generated gadgets used for correctness checks). The improvement came from a number of system-level improvements together with a branch-and-bound strategy to prune many of the $\Omega(k^m)$ constraints. We discuss this in more detail in Section 3.2.

We emphasize that our theorems do not depend on the correctness of these fast verifiers; even though our AlphaEvolve computations internally use them, the gadgets found by AlphaEvolve were verified by a brute-force algorithm that explicitly checks all $\Omega(k^m)$ constraints. In Section 5 we argue that these constructions are unlikely to have been found by hand, through conventional computational techniques such as SAT solvers, or by "directly prompting" a language model (the method most widely reported in the literature [] as evidence that AI can help research in mathematics). Instead our approach here is to invoke AlphaEvolve to autonomously find and verify the structures used in these proofs. In part we are unable to assess the robustness of direct prompting techniques because – to date – in many cases it is unclear how much human effort went into "prompt engineering" the model, relative to the contribution of the model. We are unable to reproduce our AlphaEvolve-derived results through a fair amount of such prompt engineering.

2 Hardness of Certification Problems on Random Graphs

A central problem in average case complexity is that of *certification*, where our goal is to certify a property of typical samples from a distribution. We focus on the setting of *sparse random graphs*, where the distribution in question is $\mathcal{G}(n, d)$, the uniform distribution over all n-vertex d-regular graphs, where $d \in \mathbb{N}$ is small. Let Ω be the set of n-vertex graphs, and consider a *property* $P : \Omega \to \{0, 1\}$ such that $\mathbf{Pr}_{G \sim \mathcal{G}(n,d)}[P(G) = 1] = 1 - o_n(1)$. We are given as input a graph $G \sim \mathcal{G}(n, d)$, and our algorithmic task is to efficiently certify that G satisfies P with high probability.

Definition 2.1. *An efficient certificate of P is any property $P' : \Omega \to \{0, 1\}$ that is computable in polynomial time such that the following conditions hold.*

1. *$P'(G) = 1$ implies $P(G) = 1$.*

2. $\mathbf{Pr}_{G\sim\mathcal{G}(n,d)}[P'(G)=1] = 1 - o_n(1)$.

Here, $\mathcal{G}(n,d)$ is the uniform distribution over all n-vertex d-regular graphs for some fixed constant $d \in \mathbb{N}$.

We focus specifically on the setting of *refutation*, where P is a co-NP property (recall that a problem is in co-NP if the NO instance of a decision problem has an efficiently verifiable certificate). This setting has seen considerable activity, leading to innovation in various algorithmic techniques such as spectral and sum-of-squares algorithms, along with the development of a variety of methods to establish hardness (for instance, [BHK$^+$19, RRS17, BBK$^+$21, KY24, KVWX23]).

In this work, we consider the problem of certifying upper bounds on the MAX-CUT (or MAX-IND-SET) of a graph drawn from $\mathcal{G}(n,d)$. For example, for MAX-CUT, we study the following question: for what values of σ can we efficiently certify with high probability that a random graph drawn from $\mathcal{G}(n,d)$ has MAX-CUT at most $\sigma \times \#(\texttt{of edges})$?

Definition 2.2 (Max-Cut fraction, Max-Independent Set fraction)**.** *For an undirected graph G, $\mathrm{MC}(G)$ is defined as the fraction of edges in a maximum cut of G. Similarly, $\mathrm{IS}(G)$ is defined as the fraction of vertices in a maximum independent set of G.*

Definition 2.3. *σ_d^{MC} is the infimum over all σ such that the property $\mathbf{1}\{\mathrm{MC}(G) \le \sigma\}$ has an efficient certificate. Similarly, σ_d^{IS} is the infimum over all σ such that $\mathbf{1}\{\mathrm{IS}(G) \le \sigma\}$ has an efficient certificate.*

There is evidence that this is a nontrivial problem exhibiting a *certification gap*, in the sense that $\mathrm{MC}(G)$ concentrates around a constant value strictly smaller than σ_d^{MC} for $G \sim \mathcal{G}(n,d)$ [BBK$^+$21, KY24]. These represent the structurally most elementary refutation-style properties (involving only binary variables and pairwise constraints) that exhibit interesting complexity-theoretic phenomena. In this work, we focus on accurately characterizing the value of σ_d^{MC} and σ_d^{IS}. We remark that the asymptotics of these quantities as $d \to \infty$ are already understood (thanks to [BBK$^+$21, KY24]). In service of a precise understanding of this problem in all regimes, we focus on the least understood case where d is small; specifically we study $d \in \{3,4\}$.

Towards this goal, [KY24] introduced a conjecture[3] that relates σ_d^{MC} and σ_d^{IS} to a fundamental question in spectral graph theory, specifically about Ramanujan graphs (Definition 2.4 below), which are (in a spectral sense) the most "random" looking regular graphs constructed via a deterministic process.

The intuition behind this framework relies on the strategy of "quiet planting". Specifically, if one can exhibit a Ramanujan graph failing to satisfy a certain property P (e.g., a small MAX-CUT value), then under the hardness assumption of [KY24, Conjecture 1.6], it is impossible for a polynomial-time algorithm to certify P on the random graph distribution $\mathcal{G}(n,d)$. In the context of MAX-CUT, if we construct a Ramanujan graph with a maximum cut of at least $\gamma_d^{\mathrm{MC}} \times \#(\texttt{edges})$, then the certification bound σ_d^{MC} must essentially be at least γ_d^{MC}, where γ_d^{MC} is defined in Definition 2.5 below. Therefore, the objective in establishing stronger lower bounds for certification is to maximize this value γ_d^{MC} over the set of Ramanujan graphs. We employ AlphaEvolve to construct such extremal Ramanujan graphs for both MAX-CUT and MAX-IND-SET. The precise conjecture we operate with is stated in Conjecture 2.1.

Definition 2.4 (Ramanujan graphs)**.** *Let G be a d-regular multigraph on n vertices with adjacency matrix A. Suppose the eigenvalues of A are $\lambda_1 \ge \lambda_2 \ge \ldots \ge \lambda_n$. We say that G is Ramanujan if $\max_{i>1} |\lambda_i| \le 2\sqrt{d-1}$.*

Definition 2.5. *Define γ_d^{MC} (resp. γ_d^{IS}) as the supremum of $\mathrm{MC}(G)$ (resp. $\mathrm{IS}(G)$) over all d-regular Ramanujan graphs G.*

γ_d^{MC} (resp. γ_d^{IS}) can be thought of as the performance of the optimal "spectral" certificate for bounds on MAX-CUT (resp. MAX-IND-SET).

Conjecture 2.1 ([KY24])**.** *$\sigma_d^{\mathrm{MC}} \ge \gamma_d^{\mathrm{MC}}$ and $\sigma_d^{\mathrm{IS}} \ge \gamma_d^{\mathrm{IS}}$ for all $d \ge 3$.*

[3] We refer the reader to [KY24] for supporting evidence and intuition.

	LB ([KY24])	LB (Theorem 2.1)	UB (Theorem 2.2)	UB ([Hof03, Hae21])
MAX-CUT, $d = 3$	0.944	0.944	**0.953**	0.971
MAX-CUT, $d = 4$	0.875	**0.911**	**0.916**	0.933
MAX-IND-SET, $d = 3$	0.458	**0.472**	**0.476**	0.485
MAX-IND-SET, $d = 4$	0.428	**0.454**	**0.457**	0.464

Table 1: Comparison of hardness (LB) and algorithms (UB) for certifying upper bounds on MAX-CUT and MAX-IND-SET for $\mathcal{G}(n, d)$, in terms of lower bounds and upper bounds on σ_d^{MC} or σ_d^{IS} (see Definition 2.3). The LB results are conditional on Conjecture 2.1. Our improvements are **highlighted**.

[KY24] exhibited finite d-regular Ramanujan graphs for $d \in \{3, 4\}$, yielding concrete lower bounds on σ_d^{MC} and σ_d^{IS} that are strong enough to prove a certification gap. On the other hand, there was a large gap between the best known upper bound [Hof03, Hae21] and their lower bounds (see Table 1 for a comparison). Our results make progress towards a more precise understanding of this question. Firstly, we use AlphaEvolve to construct better d-regular Ramanujan graphs, exhibiting stronger lower bounds.

Theorem 2.1 (Lower bound)**.** *We have the lower bounds* $\gamma_4^{\mathrm{MC}} \geq 113/124$, $\gamma_3^{\mathrm{IS}} \geq 17/36$, *and* $\gamma_4^{\mathrm{IS}} \geq 74/163$. *Under Conjecture 2.1, this implies the following computational hardness results:* $\sigma_4^{\mathrm{MC}} \geq 113/124 > 0.911$, $\sigma_3^{\mathrm{IS}} \geq 17/36 > 0.472$, *and* $\sigma_4^{\mathrm{IS}} \geq 74/163 > 0.453$.

This Theorem improves upon the results of [KY24], who prove $\gamma_4^{\mathrm{MC}} \geq 7/8$, $\gamma_3^{\mathrm{IS}} \geq 11/24$, and $\gamma_4^{\mathrm{IS}} \geq 3/7$. An explicit description of the constructions found by AlphaEvolve can be found in Appendix B.1, and a figure of the construction achieving the γ_4^{MC} bound is presented in Figure 1. We complement these hardness results with an algorithmic improvement over Hoffman's classic bound [Hof03, Hae21].

Theorem 2.2 (Upper bound)**.** *We have* $\sigma_3^{\mathrm{MC}} \leq 0.953$, $\sigma_4^{\mathrm{MC}} \leq 0.916$, $\sigma_3^{\mathrm{IS}} \leq 0.476$, *and* $\sigma_4^{\mathrm{IS}} \leq 0.457$.

Proof ideas for the MAX-CUT *upper bound.* Similar to existing bounds, we will use the spectral bound $\max_{i>1} |\lambda_i| \lesssim 2\sqrt{d-1}$, that is satisfied with high probability when $G \sim \mathcal{G}(n, d)$. Here $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ are the eigenvalues of the adjacency matrix $\mathbf{A}$ of G.

Consider a cut $(S, \bar{S})$ in G such that α fraction of edges cross $(S, \bar{S})$. Hoffman's bound proceeds by constructing the vector $\mathbf{x} = (2 \cdot \mathbf{1}\{i \in S\} - 1, i \in [n])$. The spectral bound above implies an upper bound on $|\mathbf{x}^\top \mathbf{A}\mathbf{x}|$, which in turn constrains α. We generalize this approach by considering inequalities involving new higher-order quadratic forms of the form $\mathbf{x}^\top \mathbf{A}^L \mathbf{x}$, where $L \in \mathbb{N}$. It is much less clear how bounds on $\mathbf{x}^\top \mathbf{A}^L \mathbf{x}$ imply better bounds on α; we show that the strongest bound one can derive on α can be captured by a linear program over probability distributions on $\{\pm 1\}$-labelings of a d-regular tree of depth L. Such a distribution is meant to model the local view of $(S, \bar{S})$ experienced by a random vertex. We solve this LP for small fixed values of $L > 1$, finding that one can improve on Hoffman's bound (corresponding to $L = 1$). □

We prove this result in Appendix B.2. For reference, we have collected our results and how they compare to previous results in Table 1. Notably, we obtain upper and lower bounds on σ_4^{MC}, σ_3^{IS}, and σ_4^{IS} that match up to a small absolute error of 0.005, so both Theorems 2.1 and 2.2 are nearly optimal for these cases. In particular, this raises the exciting possibility that one might be able to obtain clues toward complete characterizations of σ_d^{MC} and σ_d^{IS} by studying the proofs of Theorems 2.1 and 2.2.

Comments on finding near-optimal lower bounds with AlphaEvolve. We conclude this section with remarks on our methodology in Theorem 2.1, and our results. The constructions given in [KY24] for $d \in \{3, 4\}$ are graphs on up to 12 vertices that were generated by computer-assisted experimentation. We were able to replicate their results by randomly sampling a large number of regular graphs, albeit with the post-hoc knowledge of the number of self-loops in their construction.

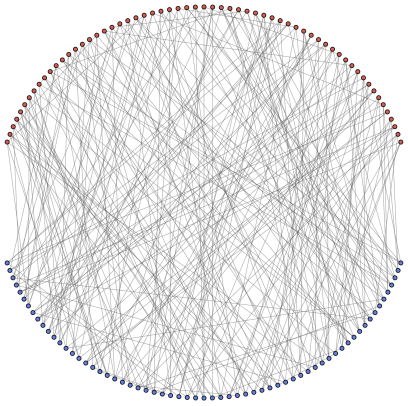


Figure 1: 4-regular Ramanujan graph found by AlphaEvolve for the lower bound on γ_4^{MC} in Theorem 2.1.

Improving their lower bounds necessitates constructions on many more vertices, as the granularity of $\mathrm{MC}(G)$ or $\mathrm{IS}(G)$ is limited by the size of G. At our target scale, the approach of randomly sampling constructions does not work for two reasons: (1) the space of d-regular n-vertex graphs blows up combinatorially, meaning random sampling will not find interesting "extremal" graphs, and (2) the complexity of computing $\mathrm{MC}(G)$ or $\mathrm{IS}(G)$ is exponential in n, so it is hard to even compute these bounds for a construction.

As stated before, we used AlphaEvolve as a tool to search for finite Ramanujan graphs witnessing the bounds in Theorem 2.1. We circumvent the above limitation by searching over the space $\mathcal{S} = \bigcup_{n=1}^{\infty} \Omega_n \times 2^{[n]}$, where Ω_n is the set of n-vertex d-regular graphs. That is, we search directly for a pair (G, S), where G is a graph and S is a cut (or independent set) in G, scoring the pair as $\texttt{score}(G, S) = -\infty$ if G is not Ramanujan, and $\texttt{score}(G, S) = |\delta_G(S, \bar{S})|/|E(G)|$ otherwise. Here, $\delta_G(S, \bar{S})$ is the set of edges crossing the cut $(S, \bar{S})$ in G. This gives an efficiently computable score function that lower bounds γ_d^{MC} (or γ_d^{IS}). Consequently, AlphaEvolve isn't limited to the space of small graphs. Indeed, it outputs constructions on as many as 163 vertices (for γ_4^{IS}).

For $d = 3$ for MAX-CUT, AlphaEvolve recovered the existing bound $\gamma_3^{\mathrm{MC}} \geq 17/18$ of [KY24], but wasn't able to improve on it. We note that for the sake of conciseness we restricted ourselves to the $d \in \{3, 4\}$ cases. We expect our techniques to achieve similar improvements over known bounds on γ_d^{MC} and γ_d^{IS} ([BBK$^+$21, Hof03]) for all reasonably small values of $d > 4$ except when $d - 1$ is an odd perfect square, when [BBK$^+$21] already gives the optimal lower bound.

3 Gadget based NP-Hardness for Approximating MAX-k-CUT

The field of *approximation algorithms* [WS11] concerns finding approximately optimal solutions to computationally hard combinatorial optimization problems. In *hardness of approximation* [AB09] the main goal is to understand when such approximations are computationally hard. We work within the well-studied framework of Constraint Satisfaction Problems (CSPs).

Consider a collection of predicates $\mathcal{P} = \{\Gamma_1, \ldots, \Gamma_r\}$ over some finite alphabet Ω, where the predicates are functions $\Gamma_i : \Omega^{l_i} \to \{0, 1\}$. A CSP is defined by a set of variables and a collection of clauses, where each clause applies a predicate from $\mathcal{P}$ to a subset of the variables, and the goal is to find an assignment of values from Ω to the variables that maximizes the number of satisfied constraints. The hardness of approximating a CSP is parameterized by two parameters c (completeness) and s (soundness). In this paper we will prove statements of the following form: Assuming $\texttt{P} \neq \texttt{NP}$, it is hard to distinguish between CSP instances where the maximum fraction of satisfied clauses is at least c, from those where it is at most s. This in turn implies that it is $\texttt{NP}$-hard to approximate the maximum number of satisfied clauses in a CSP instance within a factor

of s/c. In this paper we will parameterize the hardness of CSPs with the tuple (c, s).

Definition 3.1 (CSP)**.** *Let Ω be a finite alphabet, and let $\Gamma_1, \ldots, \Gamma_r$ be a collection of predicates over Ω, that is, $\Gamma_i : \Omega^{\ell_i} \to \{0,1\}$ for some $\ell_i \in \mathbb{N}$. We define $\mathrm{CSP}(\Gamma_1, \ldots, \Gamma_r)$ to be the collection of instances $\mathtt{I} : \Omega^n \to \mathbb{R}_{\geq 0}$ of the form $\mathtt{I}(\mathbf{x}) = \sum_{j \in [m]} \zeta_j(\mathbf{x})$ where for each j, ζ_j is a clause, i.e., it takes the form $\zeta_j(\mathbf{x}) = \Gamma_i(x_{a_1}, x_{a_2}, \ldots, x_{a_{\ell_i}})$ for some $i \in [r]$ and distinct $a_1, \ldots, a_{\ell_i} \in [n]$.*

In the Max-CSP setting, the input is a description of an instance I of a particular CSP, and the goal is to compute the *maximum* number of constraints that can be satisfied. In other words, we are trying to approximately compute the function $f(\mathtt{I}) = \max_{\mathbf{x} \in \Omega^n} \mathtt{I}(\mathbf{x})$. It will be useful to cast the approximation problem as a decision problem, as we do below by introducing the notion of (c, s)-approximation.

Definition 3.2 ((c,s)-approximation, α-approximation)**.** *Given a constraint satisfaction problem $\mathrm{CSP}(\Gamma_1, \ldots, \Gamma_r)$ and constants $0 \leq s \leq c \leq 1$, the (c, s) approximation problem for $\mathrm{CSP}(\Gamma_1, \ldots, \Gamma_r)$ is the problem of, given an n-variable instance $\mathtt{I} = \sum_{j \in [m]} \zeta_j$, deciding whether (1) $\max_{\mathbf{x}} \mathtt{I}(\mathbf{x}) \geq c \cdot m$ or (2) $\max_{\mathbf{x}} \mathtt{I}(\mathbf{x}) \leq s \cdot m$. Similarly for $0 < \alpha \leq 1$, the α-approximation problem is the problem of finding some $\mathbf{x}'$ such that $\mathtt{I}(\mathbf{x}') \geq \alpha \cdot \max_x \mathtt{I}(\mathbf{x})$.*

In particular, an α-approximation algorithm for the Max-CSP problem, with $\alpha \geq s/c$, gives an (c, s)-approximation algorithm. The main driving question can be stated as "given a CSP, what are all pairs $0 \leq s \leq c \leq 1$ such that there is an efficient (c, s) approximation for the CSP?"

We have a strong quantitative understanding for a number of CSPs [Hås01, Cha16] under NP-hardness. However, there are some quite simple CSPs for which very little is known quantitatively. For example, consider MAX-k-CUT, the problem of partitioning the vertices of a graph into k sets in a way that maximizes the number of edges crossing the partition. This problem can be written as a CSP over alphabet $\Omega = \mathbb{Z}_k$ as MAX-k-CUT $:=$ $\mathrm{CSP}(\mathtt{P}_2^{\neq})$ where $\mathtt{P}_2^{\neq}$ is the "inequality" predicate $\mathtt{P}_2^{\neq}(x, y) = \mathbf{1}\{x \neq y\}$. For small values of k, there is a large gap between the best known algorithmic results (concretely, 0.836 and 0.857-approximations for $k = 3$ and $k = 4$ respectively [GW01, dKPW04]) and the best known NP-hardness ($16/17 + \varepsilon \approx 0.941$ and $85/86 + \varepsilon \approx 0.9883$ for $k = 3$ and $k = 4$ respectively [AOTW14])[4].

Gadget reductions provide a methodical framework to find reductions between problems. Specifically in the context of inapproximability, let $\mathrm{CSP}_{\text{source}}$ be a problem with known inapproximability, and suppose we would like to prove inapproximability for another problem $\mathrm{CSP}_{\text{target}}$. The idea is to systematize the set of reductions from $\mathrm{CSP}_{\text{source}}$ to $\mathrm{CSP}_{\text{target}}$ by considering only "local" reductions that replace each clause of an instance of $\mathrm{CSP}_{\text{source}}$ with a collection of new variables, along with an instance of $\mathrm{CSP}_{\text{target}}$, known as a *gadget* on the variables involved. Crucially, the gadget for a particular predicate of $\mathrm{CSP}_{\text{source}}$ is a finite instance of $\mathrm{CSP}_{\text{target}}$ of a fixed constant size.

As an illustrative example, consider the reduction from 3-SAT to MAX-2-SAT appearing in [TSSW00]. Here, the source constraint is a 3-CNF clause $C = x_1 \vee x_2 \vee x_3$. This single constraint is mapped to a gadget consisting of a collection of ten 2-CNF clauses. The local assignment is defined over the *original variables* $\{x_1, x_2, x_3\}$ along with an *auxiliary variable* z introduced specifically for this gadget.

The target collection contains the singleton clauses x_1, x_2, x_3, z, the pairwise negated clauses of the variables, and mixed clauses linking the variables to the auxiliary. The full set of target clauses is:

$$x_1, x_2, x_3, \neg x_1 \vee \neg x_2, \neg x_2 \vee \neg x_3, \neg x_3 \vee \neg x_1,$$
$$z, \neg x_1 \vee \neg z, \neg x_2 \vee \neg z, \neg x_3 \vee \neg z.$$

In this construction, if the source constraint is satisfied, there exists a local assignment (a value for z) such that a fraction $c = 0.7$ (7 out of 10) of the target clauses are satisfied. Conversely, if the source constraint is not satisfied, no local assignment can satisfy more than a fraction $s = 0.6$ of the target constraints.

We use AlphaEvolve to find such gadget reductions from the well-understood CSP 3LIN(k) to MAX-k-CUT for $k \in \{3, 4\}$. Here 3LIN(k) is defined as CSP $\left(\mathtt{P}_{3,k}^{\equiv 0}, \mathtt{P}_{3,k}^{\equiv 1}, \ldots, \mathtt{P}_{3,k}^{\equiv (k-1)}\right)$ for the 3-ary "linear equation" predi-

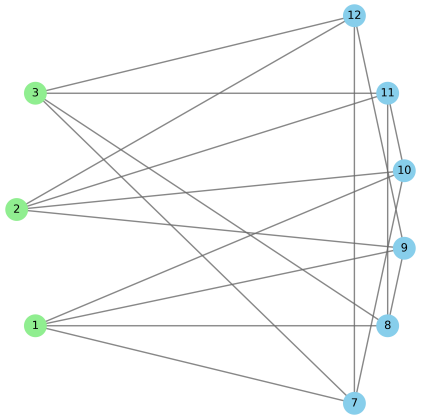
[4]Even under the Unique Games Conjecture, the optimal approximability curve is not known for $k = 4$, and for $k = 3$ is only known in the low completeness regime [Hei23] (which implies a weak inapproximability of 0.989).

cates $\mathtt{P}^{\equiv i}_{3,k}(x,y,z) = \mathbf{1}\{x+y+z \equiv i \pmod k\}$. We adopt the notation $\mathtt{P}^{\mathtt{type}}_{r,k}$, where the superscript specifies the nature of the constraint, r denotes the arity, and k (if present) indicates the alphabet size.

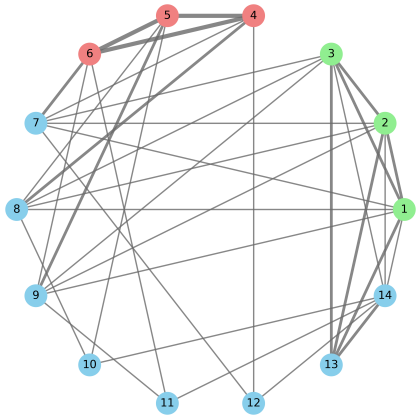
Theorem 3.1. *For any $\varepsilon > 0$, there is a gadget reduction from $(1-O(\varepsilon), 1/3+O(\varepsilon))$-approximating* 3LIN(3) *to $(0.9193-\varepsilon, 0.887+\varepsilon)$-approximating* MAX-3-CUT. *As a consequence, it is NP-hard to $0.9649+\varepsilon$-approximate* MAX-3-CUT. *More precisely, the completeness and soundness correspond to $(57/62-\varepsilon, 55/62+\varepsilon)$, and the corresponding to a hardness of $55/57+\varepsilon$.*

Theorem 3.2. *For any $\varepsilon > 0$, there is a gadget reduction from $(1-O(\varepsilon), 1/4+O(\varepsilon))$-approximating* 3LIN(4) *to $(0.9349-\varepsilon, 0.9238+\varepsilon)$-approximating* MAX-4-CUT. *As a consequence, it is NP-hard to $0.987+\varepsilon$-approximate* MAX-4-CUT.

We prove the results in Appendix C. In particular, we used AlphaEvolve to search for the gadgets used in these results. Since MAX-k-CUT is a symmetric binary predicate, we can view a gadget, which is an instance of MAX-k-CUT, as an undirected graph (possibly containing parallel edges). In Figures 2 and 3, we provide a visual representation of these instances. We note that our gadgets look extremely different in the $k=3$ and $k=4$ cases; in particular, in the $k=3$ case our gadgets contain at most 3 parallel copies of each edge, while in the $k=4$ case the gadgets contain as many as 1429 parallel copies of some edges. As a result, Theorem 3.2 does not have an inapproximability factor that can be written as a rational number with a small denominator.



(a) Gadget reducing $\mathtt{P}^{\equiv 0}_{3,3}$ to $\mathtt{P}^{\neq}_{2}$. The vertices $\{1,2,3\}$ are primary variables, and the rest are auxiliary variables. All edges represent single copies of $\mathtt{P}^{\neq}_{2}$ clauses. The global variables are not pictured as they are unused.



(b) Gadget reducing $\mathtt{P}^{\equiv 1}_{3,3}$ to $\mathtt{P}^{\neq}_{2}$. The vertices $\{1,2,3\}$ are primary variables, $\{3,4,5\}$ are global variables, and the rest are auxiliary variables. The edges represent one, two, and three copies of the corresponding $\mathtt{P}^{\neq}_{2}$ clause, depending on the thickness.

Figure 2: Gadgets found by AlphaEvolve for reducing 3LIN(3) to MAX-3-CUT (see Appendix C for a more explicit description via edge lists). Note that an edge (i,j) in the graph corresponds to the predicate $\mathtt{P}^{\neq}_{2}$ applied to variables i and j, with thickness proportional to its number of copies in the instance.

Comparison to prior inapproximability results. To the best of our knowledge, the current best hardness factor for MAX-4-CUT is $85/86+\varepsilon$ [AOTW14, Theorem 1.2], which we improve with Theorem 3.2 to 0.987. The hardness in [AOTW14] was obtained by a reduction from MAX-3-CUT to MAX-k-CUT, for all $k > 3$.

As for MAX-3-CUT, we compare our result in Theorem 3.1 to three works [KKLP96, GS09, AOTW14], who obtained successively stronger NP-hardness results for MAX-3-CUT. Our inapproximability ratio of

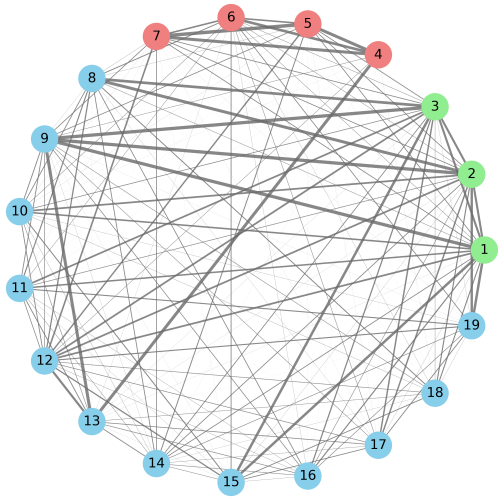


Figure 3: Gadget found by AlphaEvolve for reducing 3LIN(4) to MAX-4-CUT. The thickness of an edge (i, j) is proportional to the number of copies of the predicate $\mathtt{P}_2^{\neq}$ applied to variables i and j in the gadget, which is between 1 and 1429 (see Appendix C for a more explicit description via edge lists).

$55/57 + \varepsilon$ in Theorem 3.1 beats the $67/68 + \varepsilon$ of [KKLP96][5] and $32/33 + \varepsilon$ of [GS09].

Our result is not strong enough to beat the state of the art of $16/17 + \varepsilon$ by Austrin, O'Donnell, Tan, and Wright [AOTW14], who use a custom reduction from Label Cover. In contrast, our result does not require any new PCP machinery, utilizing only Håstad's classic PCP [Hås01]. We are unaware of a fundamental barrier that limits the gadget based approach from beating the inapproximability ratio of [AOTW14], and it is conceivable that a gadget based reduction from a different source problem to 3LIN(3) can achieve this.

3.1 Developing a Search Framework for Soundness and Completeness Guarantees

In order to apply AlphaEvolve to this problem, we developed a template for a gadget-based reduction argument and isolated the properties required from the gadget in this argument (see Definition C.1 and Theorem C.2 for details). We scored a candidate gadget by its final performance, that is, the inapproximability ratio that is implied by applying Theorem C.2 to the candidate gadget.

This template requires k separate gadgets $\{\mathtt{I}_{3,k}^{\equiv i} : 0 \leq i < k\}$, that correspond to the predicates $\{\mathtt{P}_{3,k}^{\equiv i} : 0 \leq i < k\}$ of 3LIN(k). An important feature of MAX-k-CUT is its S_k-symmetry – it is invariant under permuting the alphabet $\mathbb{Z}_k$. Unfortunately this feature is not shared by 3LIN(k), and as a result, our reductions require a way to break this symmetry of MAX-k-CUT. We achieve this by adding k new *global variables* to the standard systematization of gadgets [TSSW00].

Finding gadgets for MAX-3-CUT. AlphaEvolve found a good $\mathtt{I}_{3,3}^{\equiv 0}$ gadget on 9 variables (Figure 2a) quite quickly, mainly because the symmetries of $\mathtt{P}_{3,k}^{\equiv 0}$ work well with MAX-k-CUT when $k = 3$. In particular, this gadget did not require the aforementioned global variables. Building on ideas from the TSSW framework [TSSW00] we can show that this is the optimal $\mathtt{I}_{3,3}^{\equiv 0}$ gadget in terms of the final inapproximability ratio.

In contrast, finding good $\mathtt{I}_{3,3}^{\equiv 1}$ and $\mathtt{I}_{3,3}^{\equiv 2}$ gadgets required significantly more effort and some new ideas. Unlike $\mathtt{I}_{3,3}^{\equiv 0}$, we found that increasing the number of auxiliary variables consistently led to better performing

[5]This result can be deduced by plugging in the "state of the art" NP-hardness of approximation for MAX-2-CUT [TSSW00] into their argument.

gadgets, with the only bottleneck being the runtime of computing the performance of a candidate gadget (we comment more on this point in Section 3.2).

The $\mathtt{I}_{3,3}^{\equiv 1}$ gadget we use in the proof of Theorem 3.1 (Figure 2b) requires 14 total variables. We have some limited evidence that our $\mathtt{I}_{3,3}^{\equiv 1}$ gadget is "locally" optimal in the sense that it gives the optimal inapproximability result among all gadgets on 14 variables with the same witness as $\mathtt{I}_{3,3}^{\equiv 1}$ for all but one satisfying assignment $\mathbf{x}$ of $\mathtt{P}_{3,k}^{\equiv 1}$ (see the completeness case of Definition C.1 for what is meant by a "witness")[6].

All gadgets found for MAX-3-CUT had between 0 and 3 copies of each edge. This is expected, as optimal gadgets found in many previous results (e.g., [TSSW00, HHM$^+$17]) contain small integer weights. As a result, we get a clean inapproximability ratio of $55/57$.

Finding gadgets for MAX-4-CUT. The main difference in our process of finding gadgets for MAX-4-CUT was that we had to search over gadgets with more variables (as many as 19), so we required an extremely well-performing implementation of the verifier (again, we elaborate in Section 3.2). Even with this optimized verifier, evaluations were quite slow, requiring on the order of one second to evaluate a single gadget on 19 variables. As a result, it took a lot longer for AlphaEvolve to find a search algorithm that, given this optimized verifier, could find *any* nontrivial gadget. The final search algorithm AlphaEvolve produced had the distinctive property of searching over weighted gadgets with real-valued weights, resulting in gadgets with a wide variety of weights between 1 and 1429 after appropriate scaling and rounding.

3.2 Faster Verification via AlphaEvolve to Explore Larger Gadgets

The main challenge with finding large gadgets is that the cost of scoring a gadget scales exponentially in the number of variables; computing the completeness and soundness parameters described in Definition C.1 essentially amount to solving an instance of MAX-k-CUT, which requires exponential time. Even for $k = 3$, AlphaEvolve slows down significantly when searching for gadgets of size as few as 11 with a brute force MAX-3-CUT verifier.

This problem does not have an off-the-shelf solution in the form of existing fast verifiers; it is unlikely that existing SMT/MIP solvers [ES03, MML14] can be repurposed to solve MAX-k-CUT. To solve this issue, we used AlphaEvolve itself to speed up a naive brute force implementation of MAX-k-CUT, scoring a candidate implementation by runtime and correctness. In order to calculate the runtime, we created a synthetic dataset of a wide variety of MAX-k-CUT instances, drawn from 20 random models with varying amounts of planted structure. We then tasked AlphaEvolve to maximize the number of variables m for which the verifier requires at most one second on average to solve instances of size m from our dataset.

The biggest challenge was ensuring that AlphaEvolve does not cheat and find an *incorrect* verifier that is much faster. As mentioned before, we achieved correctness by (1) checking that the verifier is correct on our synthetic dataset, and (2) using a separate judge LLM to certify that a candidate verifier is correct. Each of these techniques was individually too lenient to avoid incorrect verifiers, but we found by human inspection that they were enough in combination.

We note that once m is large enough, it is not possible to label our dataset with the "ground truth" scores using a brute-force implementation (which is guaranteed to be correct). Instead, we inductively rely on the correctness of previous verifiers produced by AlphaEvolve (which have already passed the above correctness checks) that are fast enough to provide labels for large m.

We did this separately for the $k = 3$ and $k = 4$ cases, obtaining verifiers that are optimized to each particular problem. We found that systems-level improvements were most important to the $k = 3$ case, with the final verifier offloading the main $O(3^m)$ time computation to a highly optimized tensor contraction operation in `numpy`, and only performing slow python-based computation for $O(3^{m/3})$ time. This verifier resulted in a $10,000\times$ speedup for instances of size $m = 14$.

[6]Such a gadget giving the optimal inapproximability result can be quantified as a mixed integer program, which we solved computationally to determine that our gadget is the optimal one. See Section 3.3 for more details.

For $k = 4$, the final verifier used a more sequential branch-and-bound strategy along with some systems-level improvements. At $m = 19$, this provided a $10,000\times$ speedup against even a `numba`-accelerated [LPS15] brute force verifier.

3.3 Comparison to other computational techniques

We now survey some other computational techniques to find gadgets and discuss why they appear to be infeasible at the scale of our specific problems, even for the simpler setting of MAX-3-CUT.

The most straightforward comparison is the TSSW framework [TSSW00], which casts the task of finding the optimal gadget as a linear program (LP). The main difficulty in doing this is the presence of existential quantifiers in computing the completeness of the gadget. In order to eliminate these quantifiers, [TSSW00] canonicalize the auxiliary variables in the gadget. As a result, the size of the LP encoding the optimal gadget is doubly exponential in the number of satisfying assignments of the source predicate; this is 3^{3^6} for a reduction from 3LIN(3) to MAX-3-CUT, which is computationally infeasible. Sometimes (as is the case for our $\mathtt{I}^{\equiv 0}_{3,3}$ gadget for MAX-3-CUT) it is possible to argue that not all auxiliary variables are required, leading to a more tractable LP, but it is not clear that this is possible outside of very special cases.

If one wants to fix the number of variables in the gadget to a particular constant smaller than 3^{3^6} (like 14 for our gadgets), it is possible to write a mixed integer program (MIP) instead of an LP by encoding the existential constraints using integer variables. For example, solving the MIP took 10 hours even with all but one existential constraint eliminated[7]. (We used the SCIP solver [BBB+24a, BBB+24b], with a direct encoding of the problem.) Consequently the MIP approach also seems infeasible with SOTA solvers.

4 NP-Hardness of Approximating Metric TSP

The Traveling Salesman Problem (TSP) [WS11] is one of the most studied problems in combinatorial optimization. While there are many variants of this problem [KKOG21, SV14, STV20, TV19], we specifically focus on the metric variant of the problem [KKOG21], where the objective is to find a minimum weight Hamiltonian cycle in a weighted, complete, undirected graph with the weights satisfying the triangle inequality. This in turn corresponds to outputting a permutation over the vertices in the graph that has the minimum weight cycle. In this paper we are concerned with the hardness of approximating metric TSP. The best approximation algorithm achieves a factor of $1.5 - 10^{-34}$ [GKL23]. A series of results [PY93, Eng99, BHK+00, PV06, Lam14, KLS15, CC20] led up to the SOTA NP-hardness of approximation to a factor of $117/116$. We use AlphaEvolve to improve this result to $111/110$.

Theorem 4.1. *For any $\varepsilon > 0$, it is NP-hard to approximate the metric TSP to within $111/110 - \varepsilon$.*

A sparse CSP instance: We follow the framework of [Lam14, KLS15, CC20] and start with an NP-hard CSP we call Hybrid-3LIN(2), which we reduce to metric TSP to obtain Theorem 4.1. Hybrid-3LIN(2) is a particularly structured weighted CSP where every variable appears in exactly *three* equations, where the equations are either 3LIN(2) constraints of the form $x \oplus y \oplus z = b$, or binary linear constraints of the form $x \oplus y = b$, for some $b \in \{0, 1\}$. One can show that given a Hybrid-3LIN(2) instance, it is NP-hard to decide whether (1) most equations can be satisfied, or (2) one cannot satisfy a significant fraction of the equations. (This is the same source problem in [CC20], and a formal statement about the hardness of approximation is provided in Theorem D.1.)

A *spanning tour* in a graph is defined as a closed walk that visits each vertex at least once. It is convenient to work with an equivalent formulation of metric TSP [KLS15] (in terms of inapproximability), which we denote by MWST. The goal there is to find the minimum weight spanning tour in a weighted, undirected graph $G = (V, E, w)$. [CC20] first reduces from an instance of 3LIN(2) to an instance of Hybrid-3LIN(2) via

[7] This experiment was done with Krishnamurthy (Dj) Dvijotham, Google DeepMind.

an expander graph based CSP-to-CSP reduction. Then they reduce from Hybrid-3LIN(2) to MWST. Intuitively, the equivalence stems from the triangle inequality, which guarantees that any closed walk visiting vertices multiple times can be "short-cut" to form a valid Hamiltonian cycle of equal or lesser cost.

This MWST formulation is particularly advantageous for hardness reductions. By shifting the focus from finding a simple cycle to finding a minimum-cost connected Eulerian subgraph, one avoids the cumbersome task of explicitly constructing the metric closure. This allows the reduction to define weights locally on the sparse graph—typically assigning small weights to edges and large penalties to non-edges—while ensuring that the structural properties of the original hard instance are preserved.

We use AlphaEvolve to find a gadget (called the equation gadget) that improves the reduction from Hybrid-3LIN(2) to MWST, while keeping the rest of the arguments intact. This immediately improves the hardness-of-approximation factor from 117/116 to 111/110.

Reducing the sparse CSP to MWST: In the gadget reduction from Hybrid-3LIN(2) to MWST each variable is replaced by a vertex, and each equation is replaced by a corresponding gadget. An *equation gadget* (more concretely described below) gives a way to encode an assignment of `True` / `False` to the variables in a 3LIN(2) equation (i.e., clauses of the form $\mathtt{P}^{\equiv 0}_{3,2}$ or $\mathtt{P}^{\equiv 1}_{3,2}$) into a connectivity property for the corresponding vertices, in the sense that if the equation is satisfied, one can connect the involved vertices to the rest of the graph with low cost; on the other hand, if the equation is unsatisfied, then any spanning tour must dedicate a large total weight of edges in order to connect the vertices involved to the rest of the graph.

Using AlphaEvolve, we find a new equation gadget (Figure 4) achieving better performance than the one appearing in [CC20] (Figure 8a)[8]. In order to quantify this improvement, we now describe equation gadgets more concretely: each 3LIN(2) equation $\mathtt{P}^{\equiv 1}_{3,2}$ (of the form $x \oplus y \oplus z = 1$) is assigned an equation gadget, where the green vertices $\{1, 2, 3\}$ in Figure 4, also called the *contact vertices*, correspond to the variables $\{x, y, z\}$. The red vertex $\{4\}$ is called the *central vertex*, and is shared across all appearances of the equation gadget across all 3LIN(2) equations in the instance. The rest of the vertices in the equation gadget (shown in blue in Figure 2) are used to ensure that there is a gap between the weight of a spanning tour due to a satisfied 3LIN(2) equation versus an unsatisfied one. The black edges (referred to as "unforced edges") are optional in any tour, while any tour is forced to take each of the red edges (referred to as "forced edges") at least once. We note that a standard trick (e.g., [KLS15]) can be used to implement the constraint of forcing certain edges to be taken within MWST. The green dashed edges are called *special edges*, and are purely used for analysis purposes, and do not appear in the actual MWST instance.

As described earlier, an equation gadget performs well if its contribution to a spanning tour is small /large for satisfied/unsatisfied clauses respectively. In what follows, we will distill this requirement into a self-contained statement about the gadget. Given an equation gadget, we consider disjoint collections of tours within it that cover every vertex. Such a collection is associated with a particular assignment of (x, y, z), where a variable is assigned `True` if and only if its corresponding contact vertex appears in the same tour as the central vertex 4. Furthermore, each tour beyond the one containing the central vertex suffers a weight penalty of one. For an assignment (x, y, z) for a 3LIN(2) equation, we will be concerned with the minimum possible total weight of any such collection associated with (x, y, z) (including any weight penalties). We note that the above description is simplified as it does not account for "dishonest" collections of tours that interact in unwanted ways with the rest of the reduction. We adopt a slightly different formalism in our formal proofs in order to handle these, deferring the discussion to Appendix D.2.

Our improvement to the equation gadget: The improved equation gadget found by AlphaEvolve (Figure 4) admits collections having total weight 10 associated with satisfying assignments for a 3LIN(2) equation, and at least 11 associated with unsatisfying assignments, as opposed to 13 and 14 respectively for the gadget in [CC20]. This immediately improves the performance of the reduction, yielding an improvement in the inapproximability ratio of MWST (and equivalently metric TSP) from 117/116 to 111/110. A description of our new gadget, the full reduction, and a proof of the resulting inapproximability ratio as a function of the equation gadget is provided in Appendix D.

[8]The 3LIN(2) equation encoded in the equation gadgets in [CC20, KLS15, Lam14] all have the form $x \oplus y \oplus z = 0$, as opposed to $x \oplus y \oplus z = 1$ in Figure 4.

While we present the equation gadget in Figure 4 corresponding to the predicate $x \oplus y \oplus z = 1$, we can get the same approximation ratio using a gadget corresponding to $x \oplus y \oplus z = 0$ (which is in the same setup of [CC20]). The gadget is however more complex. We defer its description, and a side-by-side comparison with [CC20], to Appendix D.3.

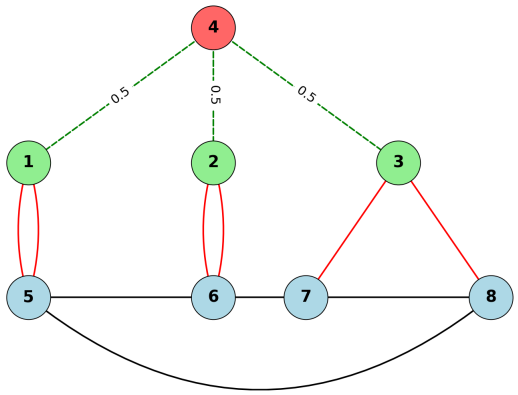


Figure 4: Equation gadget for MWST found by AlphaEvolve, when reducing from a 3LIN(2) equation $\mathrm{P}_{3,2}^{\equiv 1}$ of the form $x \oplus y \oplus z = 1$. Vertices $\{1, 2, 3\}$ represent variables in the 3LIN(2) equation. The red edges represent the forced edges. The dashed green edges represent the special edges. All red and black edges have weight one.

Core ideas, and the importance of AlphaEvolve: Of the problems discussed in this paper, TSP required the most human involvement, primarily because there was no existing search framework for gadget reductions (analogous to [TSSW00] for MAX-k-CUT). In particular, the complete reduction in [CC20] contains a scaffolding beyond the equation gadget, and all components of the reduction are analyzed together as a single object. Within the proof structure of prior work [CC20, KLS15] it seems unclear how to abstract out a set of verifiable constraints under which AlphaEvolve can be used to search for better equation gadgets alone. In this work, we modularized both the soundness and completeness proofs to depend on well-defined soundness and completeness parameters, defined as optimization problems on the equation gadgets themselves (Definition D.4). This modularization may be of independent interest for future work. We used AlphaEvolve to search for equation gadgets that maximized the final inapproximability ratio obtained from these soundness and completeness parameters.

The optimization problem for the equation gadget search can be cast as a mixed integer program (MIP). Because of its simplicity, it is perceivable that the gadget in Figure 4 can be found by directly solving the MIP assuming the number of auxiliary vertices are known in advance. However, because of the large number of constraints involved in the equation gadget corresponding to $x \oplus y \oplus z = 0$ (in Figure 8b), i.e., around 11!, using traditional SMT/MIP solvers would face the same computational bottlenecks described (in Section 3.3) for the MAX-k-CUT problem.

5 Discussion on AI-assisted Mathematics and Complexity Theory

The area of using AI for Math/CS research is vibrant and changing fast, but we approach it from a particular point of view: we seek to focus on classical and well-studied problems to derive results that stand the test of time. Employing AI currently appears to be the only known way to achieve the results in this paper. In studying the role of AI in assisting mathematical discovery, we must consider at least these scenarios:

1. We invoke a language model to summarize the state of prior art, to chart a research plan towards new theorems, or to directly generate fragments of (or entire) proofs.
2. We use AI-derived tools such as AlphaEvolve to generate better proof elements (gadgets, graphs).
3. We use custom code independent of AI to discover better proof elements.

4. We discover the same or better proof elements by hand.
5. A combination of (1)-(4).

Literature summaries: The idea that LLMs can usefully generate summaries of prior literature in a field has been mooted about for some time [WLNR23, WZLJ23, HLL$^+$24, GUK$^+$24], and indeed our experience confirmed this on a number of prompts. The results (modulo occasional hallucination) are a good starting point for deeper exploration and understanding. Interestingly, across many examples we could rapidly generate an overview of the art in an unfamiliar field; we believe this capability will increasingly be used by scientists working across fields so that — for instance — an algebraist can come up to speed with Ramsey Theory. We believe that in time, this will lead to more fluent cross-pollination across disciplines. We have not yet been able to prompt an LLM into providing a usable research plan to obtain new results (such as the ones we report), and this is the focus of deeper efforts e.g., Google's Co-Scientist [GWD$^+$25] program.

Direct Prompting: There have been efforts to generate proofs for open mathematical statements via prompting an LLM directly [Raa25, Bub25, OD25, DdMN25, JR25, BCE$^+$25]. This approach has seen mixed success. In some cases an LLM was indeed able to generate the complete and correct proof of a previously unproven statement [Bub25, BCE$^+$25] (although it is possible that a persistent human could perhaps have derived the same), but in many cases [Raa25, OD25, DdMN25, JR25] the LLM could only generate a proof sketch, which ultimately had to be filled in by humans. At the time of this writing, one significant piece of work in this category is [BCE$^+$25]. This report documents GPT-5's ability to accelerate research in mathematics and theoretical computer science, demonstrating how "scaffolding" — the technique of priming the model with simpler, related warm-up problems — enabled it to derive improved bounds for online algorithms and formally prove previously open conjectures in graph theory. The authors highlight the model's capacity to go beyond retrieval, successfully constructing novel counterexamples and proof strategies that had previously eluded human experts, such as generating a complex counterexample to the "Follow-the-Leader" algorithm in convex body chasing and proposing the "stability-style analysis" key to solving Erdős Problem #848 [Blo]. In a follow-up to our work, [WCAJ$^+$26] use Gemini to solve a large class of problems spanning math, physics, and theoretical computer science.

In general, these proofs/sketches currently require a human to verify correctness. Our own *natural attempts* to prompt a standard LLM into directly generating the kinds of combinatorial structures in this paper met with failure [9]. While this may become feasible as LLM reasoning capabilities improve [LL25], a formal comparison is beyond the current scope. We emphasize that our constructions *always come with a certificate of correctness* that is formally verified via standard computational approaches. Once the verification code is sound, there is no further need for human scrutiny.

Demonstrating AlphaEvolve's breadth: The recent paper [GGSTW25] is a sweeping tour de force of LLM-guided evolutionary search (AlphaEvolve) in autonomously discovering novel mathematical constructions across analysis, combinatorics, and geometry, surprisingly finding a counterexample to the "Four Guards" logic puzzle by creatively engaging in "prompt injection" against the verifying language model. It notably improves bounds on long-standing problems like the Kakeya needle problem and the Ring Loading Problem, often achieving results competitive with or superior to human-designed benchmarks with minimal problem-specific tuning. Importantly, in most of this work (including our gadget reductions), AlphaEvolve cannot directly "evolve the theorem"; rather, its evolutionary process is guided by a synthetic scoring function that guides towards a better theorem.

Computational methods in hardness of approximation: In both average case hardness for random graphs, and NP-hardness for approximating MAX-k-CUT, computer assisted methods [TSSW00, Hås01, KY24] have been used previously. [KY24] used computational methods to generate d-regular Ramanujan graphs (for $d \in \{3, 4\}$) with large cut values (or independent sets). While they do not specify their computational approach, we could replicate their results by random sampling of d-regular graphs, and testing for the properties by brute force. For reasons mentioned in Section 2, [KY24] could demonstrate their lower bounds for $n \leq 12$, whereas the graphs we find go up to $n = 163$.

[9] For example, GPT 5.2 and Gemini 3.0 Ultra could not generate the 16/17-NP-hardness gadget for MAX-CUT, even though the gadget is explicitly mentioned in [TSSW00, Figure 4.1] — the prompt being `Can you generate the gadget that gives the construction for 16/17 approximation of MAX-CUT under NP hardness?`

For the NP-hardness gadget reduction in Section 3, both the soundness and the completeness constraints can be translated into a linear program via skolemization [TSSW00]. However, the size of such a linear program is doubly exponential in the number of vertices in the constraint graph. As a result, even for MAX-3-CUT, running a linear program (LP) with the *canonical number of variables* [TSSW00] $\approx 3^{3^6}$ becomes infeasible with the standard LP solvers we know of. One can directly attack the non-convex program using SMT/MIP solvers, but with the number of constraints being exponential in the number of variables $n \geq 14$, they too did not seem to scale. (See Section 3.3 for a detailed discussion.) In the case of hardness of approximating the TSP, the soundness and completeness constraints can also be cast as MIP constraints. Even with a modest number of vertices in the equation gadget (e.g., twelve in Figure 8b), the number of constraints becomes prohibitively large for standard MIPs to handle (around $11! \approx 3 \times 10^7$).

Gadget design by hand: It is conceivable that a human expert or a highly customized computational verifiers could eventually find these solutions. However, their discovery is likely beyond the reach of simple "pencil-and-paper" methods, and standard SMT/MIP solvers failed to produce them. Further, humans intuitively cut the search space through insights into symmetries in the constructed objects; in the case of our TSP gadget, it appears that asymmetry was central to the improvement obtained.

AI with significant human effort: Our work and [Tao25] are instances where AI and some significant human effort were deployed in combination. In particular, [Tao25] chronicles the rapid resolution of Erdős Problem #1026 [Blo] through a hybrid workflow where human ingenuity guided multiple AI systems—including AlphaEvolve for generating optimal constructions and the automated theorem prover Aristotle [ABB+25] for formal verification in Lean. By integrating crowd-sourced mathematical insights with AI-driven deep literature search and computational discovery, the collaboration successfully generated the solution in under 48 hours. In our case, we had to refactor (by hand) the proof logic especially for metric TSP, in order for the problem to be amenable to AlphaEvolve.

In the spirit of Turing's imitation game [Tur50], one strong test of robustness for AI-assisted mathematical discovery is durability: whether or not new results obtained with AI assistance are superseded by humans (possibly with computer assistance). We note that some results in AI-assisted mathematics have in fact been matched or improved quickly without the use of AI [Ger25, BSZ25], and in some cases the results existed (even before the problem was posed [Alo24, Rec25]) but were not known to the authors when AI methods were applied [AM25, BCE+25].

Concluding remarks: While our experience here is limited and far from definitive, we believe some early themes are emerging.

First, language models can generate research plans and summarize the state of the art [GWD+25]. While we have not succeeded in deriving novel results from this, the capability allows non-specialists to quickly learn new domains, which we anticipate will foster greater scientific cross-pollination.

Second, we expect a growing number of proofs of the form "AI got there first" without clear evidence of the form "this couldn't have been done without AI". In all of these cases (and arguably, across applications of AI to science), we expect verification to be an ongoing bottleneck. We leave it as an open question whether directly prompting an LLM can eventually replicate and surpass our results.

Third, our work suggests that gadget-based reductions lend themselves to optimization beyond traditional methods (e.g., SMT/MIP solvers), using AlphaEvolve. This in turn suggests that beating AlphaEvolve will generally require non-gadget methods like custom PCPs [AOTW14].

Finally, it is worth dwelling on some failures of our approach. For some problems even if verification is trivial we could not get AlphaEvolve to work. An example of this is the Hadamard-668 conjecture which states that there exists a Hadamard matrix of dimensions 668×668. (More generally, it is conjectured that a Hadamard matrix $H_{n\times n}$ exists for any n which is a multiple of four; 668 is the smallest value for which this is not known.) We attempted to construct one using AlphaEvolve. Although the search is over a large space of 2^{668^2} possibilities, verifying any candidate for correctness only takes $2,23,112$ bitwise multiplications. Even with fast verification, AlphaEvolve was unable to find a construction for $H_{668\times 668}$. In fact, we failed to get AlphaEvolve to replicate the construction for $H_{428\times 428}$, which was previously the smallest order for which no construction was known until [KTR05]; this despite the fact that the construction for 428 is publicly

available on the internet.

Looking ahead, it is conceivable that advances in LLM reasoning [LL25, GWD+25, Wei25] could be coupled with AlphaEvolve, especially in generating the initial code-snippet and more effective problem specific prompting of the LLMs used by AlphaEvolve. We leave exploration of these problems as future directions.

6 Acknowledgments

We thank Adam Zsolt Wagner for helping us with AlphaEvolve throughout this project, and for his invaluable advice on our experimental setup — his insights speeded up our experimentation significantly. Swarat Chaudhuri not only helped us with our initial work on MAX-3-CUT, but also generously advised us throughout on various approaches to verification — we are deeply grateful to him for this. Sushant Sachdeva worked closely with us on MAX-3-CUT, TSP, and the design of verifiers and freely shared his valuable intuition on gadget reductions. Pasin Manurangsi replicated some of our early gadgets by hand (especially the *optimal* gadget that reduces from $\mathtt{P}_{3,3}^{\equiv 0}$ to $\mathtt{P}_{2}^{\neq}$), boosting our confidence early on that AlphaEvolve was progressing in the right direction. Pasin also helped us solidify our understanding of [KLS15, CC20]. Sidhanth Mohanty made critical contributions in improving the upper bounds on average-case hardness (Theorem 2.2). Krishnamurthy (Dj) Dvijotham helped us with the advanced use of MIP solvers, and ablation studies with some of our verifiers. His ablation studies convinced us that direct usage of MIP solvers would not scale to the gadget sizes we are operating with. Shuang Song helped us with numerous engineering challenges. We thank Mary Chesus, Jonathan Katz, Ravi Kumar, James Manyika, Yossi Matias, Jelani Nelson, Rina Panigrahy, Raluca-Ada Popa, Amit Sahai, Thomas Steinke and Jalaj Upadhyay for their valuable feedback on the manuscript. We thank Uri Feige for pointing out an error in a previous version of Appendix B.2. We thank Four Flynn and Pushmeet Kohli for their continued support and feedback through the course of this project.

We especially thank Venkat Guruswami and Madhu Sudan for their ongoing encouragement for this work, and for their incisive perspectives on the current status of various results in inapproximability. In particular, Venkat helped us with nuances of the state of the art for MAX-k-CUT, and pointed us to [GS09, AOTW14].

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[ABB+25] Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, et al. Aristotle: Imo-level automated theorem proving. *arXiv preprint arXiv:2510.01346*, 2025.

[Alo24] Noga Alon. Graph-codes. *European Journal of Combinatorics*, 116:103880, 2024.

[AM25] Boris Alexeev and Dustin G Mixon. Forbidden sidon subsets of perfect difference sets, featuring a human-assisted proof. *arXiv preprint arXiv:2510.19804*, 2025.

[AOTW14] Per Austrin, Ryan O'Donnell, Li-Yang Tan, and John Wright. New NP-hardness results for 3-coloring and 2-to-1 label cover. *ACM Transactions on Computation Theory (TOCT)*, 6(1):1–20, 2014.

[BBB+24a] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns,

Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. Technical report, Optimization Online, February 2024.

[BBB+24b] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. ZIB-Report 24-02-29, Zuse Institute Berlin, February 2024.

[BBK+21] Afonso S Bandeira, Jess Banks, Dmitriy Kunisky, Christopher Moore, and Alex Wein. Spectral planting and the hardness of refuting cuts, colorability, and communities in random graphs. In *Conference on Learning Theory*, pages 410–473. PMLR, 2021.

[BCE+25] Sébastien Bubeck, Christian Coester, Ronen Eldan, Timothy Gowers, Yin Tat Lee, Alexandru Lupsasca, Mehtaab Sawhney, Robert Scherrer, Mark Sellke, Brian K Spears, et al. Early science acceleration experiments with gpt-5. *arXiv preprint arXiv:2511.16072*, 2025.

[BHK+00] Hans-Joachim Böckenhauer, Juraj Hromkovič, Ralf Klasing, Sebastian Seibert, and Walter Unger. An improved lower bound on the approximability of metric TSP and approximation algorithms for the TSP with sharpened triangle inequality. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 382–394. Springer, 2000.

[BHK+19] Boaz Barak, Samuel Hopkins, Jonathan Kelner, Pravesh K Kothari, Ankur Moitra, and Aaron Potechin. A nearly tight sum-of-squares lower bound for the planted clique problem. *SIAM Journal on Computing*, 48(2):687–735, 2019.

[Blo] Thomas F. Bloom. Erdős Problems. `https://www.erdosproblems.com/`. Accessed: 2025-12-18.

[BSZ25] Guy Barzilai, Ohad Shamir, and Moslem Zamani. Are convex optimization curves convex? *arXiv preprint arXiv:2503.10138*, 2025.

[Bub25] Sebastien Bubeck. Claim: GPT-5-pro can prove new interesting mathematics. Post on X (formerly Twitter), 8 2025. The post claims GPT-5 Pro was used to improve a mathematical bound in a convex optimization paper. The original URL is not available in the screenshot.

[CC20] Miroslav Chlebík and Janka Chlebíková. Weighted amplifiers and inapproximability results for travelling salesman problem. *Journal of Combinatorial Optimization*, 2020.

[Cha16] Siu On Chan. Approximation resistance from pairwise-independent subgroups. *Journal of the ACM (JACM)*, 63(3):1–32, 2016.

[DdMN25] Charles-Philippe Diez, Luis da Maia, and Ivan Nourdin. Mathematical research with GPT-5: a Malliavin-Stein experiment, 2025.

[dKPW04] Etienne de Klerk, Dmitrii V Pasechnik, and Joost P Warners. On approximate graph colouring and max-k-cut algorithms based on the θ-function. *Journal of Combinatorial Optimization*, 8(3):267–294, 2004.

[DL09] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 371–380, 2009.

[Eng99] Lars Engebretsen. An explicit lower bound for TSP with distances one and two. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 373–382. Springer, 1999.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[Fri08] Joel Friedman. *A proof of Alon's second eigenvalue conjecture and related problems*. American Mathematical Soc., 2008.

[Ger25] Robert Gerbicz. Sums and differences of sets (improvement over AlphaEvolve). *arXiv preprint arXiv:2505.16105*, 2025.

[GGSTW25] Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

[GKL23] Leonid Gurvits, Nathan Klein, and Jonathan Leake. From trees to polynomials and back again: New capacity bounds with applications to TSP. *arXiv preprint arXiv:2311.09072*, 2023.

[GS09] Venkatesan Guruswami and Ali Kemal Sinop. Improved inapproximability results for maximum k-colorable subgraph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 163–176. Springer, 2009.

[GUK+24] Alexander Goldberg, Ihsan Ullah, Thanh Gia Hieu Khuong, Benedictus Kent Rachmat, Zhen Xu, Isabelle Guyon, and Nihar B Shah. Usefulness of LLMs as an author checklist assistant for scientific papers: Neurips'24 experiment. *arXiv preprint arXiv:2411.03417*, 2024.

[GW01] Michel X Goemans and David Williamson. Approximation algorithms for max-3-cut and other problems via complex semidefinite programming. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 443–452, 2001.

[GWD+25] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, Yossi Matias, Andrew Carroll, Kavita Kulkarni, Nenad Tomasev, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R D Costa, José R Penadés, Gary Peltz, Yunhan Xu, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Towards an AI co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.

[Hae21] Willem H Haemers. Hoffman's ratio bound. *Linear Algebra and its Applications*, 617:215–219, 2021.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[Hei23] Steven Heilman. Three candidate plurality is stablest for correlations at most 1/10. *arXiv preprint arXiv:2306.03312*, 2023.

[HHM+17] Johan Håstad, Sangxia Huang, Rajsekar Manokaran, Ryan O'Donnell, and John Wright. Improved NP-inapproximability for 2-variable linear equations. *Theory of Computing*, 13(1):1–51, 2017.

[HLL+24] Yizhen He, Yifei Liu, Jiacheng Li, Yutong Wang, Ya-Chu Chang, Yan-Tsung Chuang, Wei-Wei Tu, Bo-An Jang, Wen-Chih Peng, and Philip S Yu. Large language models in scientific discovery: A survey. *arXiv preprint arXiv:2402.13119*, 2024.

[Hof03] Alan J Hoffman. On eigenvalues and colorings of graphs. *Selected Papers Of Alan J Hoffman: With Commentary*, pages 407–419, 2003.

[JR25] Uijeong Jang and Ernest K. Ryu. Point convergence of Nesterov's accelerated gradient method: An AI-assisted proof, 2025.

[KKLP96] Viggo Kann, Sanjeev Khanna, Jens Lagergren, and Alessandro Panconesi. On the hardness of approximating max k-cut and its dual. In *ISTCS*, pages 61–67, 1996.

[KKOG21] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 22–45. ACM, 2021.

[KLS15] Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. *Journal of Computer and System Sciences*, 81(8):1665–1677, 2015.

[KTR05] Hadi Kharaghani and Behruz Tayfeh-Rezaie. A Hadamard matrix of order 428. *Journal of Combinatorial Designs*, 13(6):435–440, 2005.

[KVWX23] Pravesh Kothari, Santosh S Vempala, Alexander S Wein, and Jeff Xu. Is planted coloring easier than planted clique? In *The Thirty Sixth Annual Conference on Learning Theory*, pages 5343–5372. PMLR, 2023.

[KY24] Dmitriy Kunisky and Xifan Yu. Computational hardness of detecting graph lifts and certifying lift-monotone properties of random regular graphs. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1621–1633. IEEE, 2024.

[Lam14] Michael Lampis. Improved inapproximability for TSP. *Theory OF Computing*, 10(1):217–236, 2014.

[LL25] Thang Luong and Edward Lockhart. Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the international mathematical olympiad. *DeepMind Blog*, 2025.

[LPS15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6. ACM, 2015.

[McK] Brendan McKay. Graph formats. `https://users.cecs.anu.edu.au/~bdm/data/formats.html`. Accessed: 2025-09-08.

[MML14] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 438–445. Springer, 2014.

[NVE+25] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[OD25] Francesco Orabona and Ryan D'Orazio. New perspectives on the Polyak stepsize: Surrogate functions and negative results. *arXiv preprint arXiv:2505.20219*, 2025.

[PV06] Christos H Papadimitriou and Santosh Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006.

[PY93] Christos H Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[Raa25] Mark Van Raamsdonk. Finite entropy sums in quantum field theory. *arXiv preprint arXiv:2508.21276*, 2025.

[Rec25] Ben Recht. The fine art of crate digging, October 2025. Accessed: 2025-10-27.

[RPBN+24] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[RRS17] Prasad Raghavendra, Satish Rao, and Tselil Schramm. Strongly refuting random CSPs below the spectral threshold. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 121–131, 2017.

[STV20] Ola Svensson, Jakub Tarnawski, and László A. Végh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. *Journal of the ACM*, 67(4):22:1–22:53, 2020.

[SV14] András Sebő and Jens Vygen. Shorter tours by nicer ears. *Combinatorica*, 34(5):597–629, 2014.

[Tao25] Terence Tao. The story of Erdős problem #1026. `https://terrytao.wordpress.com/2025/12/08/the-story-of-erdos-problem-126/`, December 2025. Accessed: 2025-12-18.

[TSSW00] Luca Trevisan, Gregory B Sorkin, Madhu Sudan, and David P Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000.

[Tur50] Alan M Turing. Computing machinery and intelligence. 1950.

[TV19] Vera Traub and Jens Vygen. Approaching 3/2 for the s-t-path TSP. *Journal of the ACM*, 66(2):14:1–14:17, 2019.

[WCAJ+26] David P Woodruff, Vincent Cohen-Addad, Lalit Jain, Jieming Mao, Song Zuo, MohammadHossein Bateni, Simina Branzei, Michael P Brenner, Lin Chen, Ying Feng, et al. Accelerating scientific research with gemini: Case studies and common techniques. *arXiv preprint arXiv:2602.03837*, 2026.

[Wei25] Kevin Weil. An AI-powered platform that accelerates scientific discovery. `https://x.com/kevinweil/status/1962938974260904421`, Sept 2025.

[WLNR23] Zhaoyuan Wang, Hai-Chau Le, Hootan Nakhost, and R. M. Rutledge. Can large language models be an alternative to human evaluations? *arXiv preprint arXiv:2305.01937*, 2023.

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[WZLJ23] Shuai Wang, Yuhan Zhang, Lin Li, and Zhaochun Jiang. Can ChatGPT Write a Good Boolean Query for a Systematic Review? *Journal of the Medical Library Association: JMLA*, 111(3):658–663, 2023.

A AlphaEvolve as a Framework for Combinatorial Discovery

AlphaEvolve [NVE+25, RPBN+24] is an LLM-based code-mutation agent. In the context of this paper, it evolves code snippets that generate combinatorial structures, scoring generated structures by their fitness for the problem at at hand. These evolved code snippets generated novel *finite combinatorial structures* that improve results in the hardness of approximation. Previously AlphaEvolve has been used in other of scientific domains [RPBN+24, NVE+25]. In the following, we provide a self-contained description of AlphaEvolve. Additionally, we highlight the system enhancements needed to achieve the combinatorial structures in this paper.

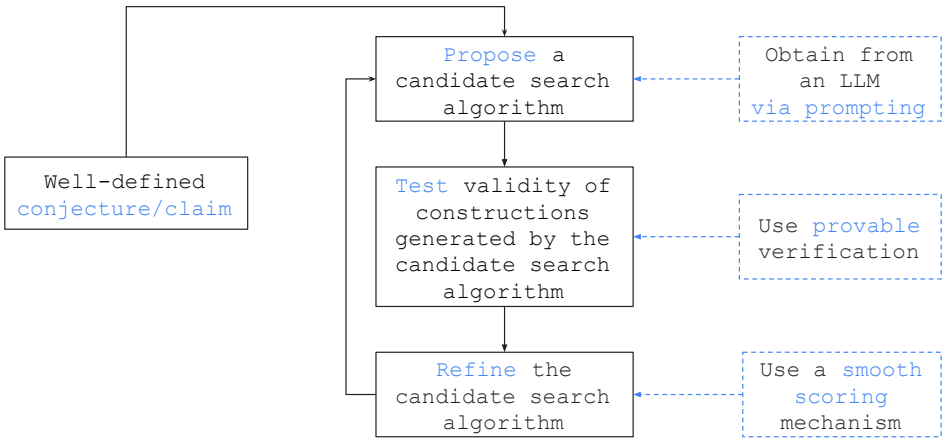


Figure 5: Propose-test-refine (PTR) paradigm: Defining combinatorial search with AlphaEvolve. The solid lines define the control flow, and the dashed lines define comments.

Propose-test-refine (PTR) paradigm: We operate in the PTR paradigm [10] for setting up experiments to discover novel combinatorial structures. While the paradigm is implicit in prior work [NVE+25], we first make it explicit, and then instantiate the extensions needed in this paper. The PTR paradigm consists of three components:

1. *Suitably defined conjecture/claim*: The first step towards discovery is a concrete definition of the combinatorial structure being sought. For example, it can be a weighted graph with some fixed set of vertices, or a SAT formula with a fixed number of clauses and variables. In this paper, we restrict these to *finite structures* rather than a parameterized family [11]. While the source and the target problems form a parameterized family (which enables us to prove theorems with $\forall n$ quantification), the gadgets used to translate between them are finite structures. (A *gadget* may be thought of as a combinatorial structure mapping a source constraint to a linear combination of target constraints [Hås01, TSSW00].)
2. *An initial candidate search algorithm*: AlphaEvolve uses an LLM to make syntactically valid modifications to a code snippet, with the goal of improving a fitness score for the combinatorial structure it generates. Denote the code snippet at time t by $\mathcal{C}_t : \mathcal{D} \to \mathcal{S}$, where $\mathcal{D}$ is the domain capturing problem specific parameters and any inputs to the snippet, and $\mathcal{S}$ is the range space of all possible combinatorial structures. Additionally, let $\texttt{score} : \mathcal{S} \to \mathbb{R}$ be the function that assigns a fitness score to a structure. The next function $\mathcal{C}_{t+1}$ is chosen as follows:

$$\mathcal{C}_{t+1} \leftarrow \texttt{AlphaEvolve}\left((\mathcal{C}_0(d_0), \texttt{score}(\mathcal{C}_0(d_0))), \ldots, (\mathcal{C}_t(d_t), \texttt{score}(\mathcal{C}_t(d_t)))\right) \tag{1}$$

   `AlphaEvolve` makes an LLM call using the tuple of inputs in the prompt to the LLM, prompting it to output the description of $\mathcal{C}_{t+1}$, with the goal of increasing the score[12]. We emphasize the following subtle point: AlphaEvolve does not directly search for combinatorial structures, but instead for *a code snippet that would output a combinatorial structure with a high score*. Figure 6 shows an initial code-snippet to be evolved by AlphaEvolve.
3. *A verifier with a smooth scoring function*: In the problems we consider, we seek combinatorial structures in spaces of size exponential in the description of the structure. A well-defined verifier that validates (and scores) the constructions generated by the $\mathcal{C}_t$'s (in Equation (1)) guides this search.

[10] The name is motivated from the Propose-test-release (PTR) framework in the differential privacy literature [DL09], which performs a similar task of testing a proposed candidate construction for (privacy) properties, before releasing it.

[11] Let $\mathcal{S}$ be a class of combinatorial structures (e.g., graphs, hypergraphs, codes, formulas). A *parameterized family of structures* from $\mathcal{S}$ is a sequence $\{S_n\}_{n\in\mathbb{N}}$ s.t. each $S_n \in \mathcal{S}$ is a finite structure whose size depends on the parameter n.

[12] Technically, AlphaEvolve does not feed in all the prior functions as the prompt to the LLM. Instead, it uses a genetic algorithm to select a subset of them while maintaining high scores, and diversity [RPBN+24, Figure 1].

```python
# EVOLVE-BLOCK-START

def gadget_construction() -> nx.graph:
  """Returns a candidate gadget construction"""
  gadget = nx.Graph()
  # Create a complete graph with random weights
  max_scoring_gadget = gadget
  max_score = evaluate_internal(gadget)
  while time.time() - start_time < 1000.0:
    # Make random modifications to gadget, to create
    # a new gadget: new_gadget
    current_score = evaluate_internal(new_gadget)
    if max_score < current_score:
      max_scoring_gadget = new_gadget
      max_score = current_score

  return max_scoring_gadget

def evaluate_internal(gadget):
  # Compute the score on the gadget, and return
  return score(gadget)

# EVOLVE-BLOCK-END
```

Figure 6: Initial code-snippet for AlphaEvolve to evolve.

Concretely, for average-case hardness in Section 2, this corresponds to validating whether the graphs are Ramanujan with certain properties (e.g., having a large cut, or independent set), and for worst-case NP-hardness in Sections 3 and 4, this corresponds to verifying whether the *soundness* and *completeness* constraints (e.g., [TSSW00, Hås01], Lemmas D.2 and D.3) for the gadget reduction are satisfied.
As mentioned earlier, every structure generated by AlphaEvolve needs to be scored, including invalid ones that violate certain constraints. The scoring function $\texttt{score} : \mathcal{S} \to \mathbb{R}$ is used to hill-climb on a optimization landscape over the combinatorial structures. For invalid structures, it decides how far from the constraint boundary the current structure lies. (Informally, a scoring function is *smooth* if its value gracefully increases/decreases as one moves away from the constraint boundary.) This is where the most creativity and problem-specific domain expertise comes in. In particular, the efficacy of the scoring function largely decides the overall capability of the system to discover novel structures. Also, scoring is usually the *most compute intensive* step in the PTR paradigm.

Our contributions: Our main contributions to the PTR paradigm are the following: a) using AlphaEvolve itself to accelerate the verification for MAX-k-CUT (*sometimes up to* $10,000\times$), and b) an approach towards using AlphaEvolve for solving optimization problems that involve mixed-integer programming (MIP) style constraints. For the $k = 3$ case, the AlphaEvolve optimized verification code constructed a verifier that offloaded a $O(3^m)$ time computation to a highly optimized tensor contraction operation in `numpy`, and only performing slow python-based computation for $O(3^{m/3})$ time. We discussed each of these contributions in detail in Sections 3.1 and 3.2.

B Proofs of Average Case Theorems

B.1 Proof of Theorem 2.1

Below we specify three Ramanujan graphs G_4^{MC}, G_3^{IS}, and G_4^{IS} in the sparse6 format [McK]. We also specify a cut S_4^{MC} (in the form of a subset of vertices), and independent sets S_3^{IS} and S_4^{IS} that witness lower bounds on the actual values of $\text{MC}(G_4^{\text{MC}})$, $\text{IS}(G_3^{\text{IS}})$, and $\text{IS}(G_4^{\text{IS}})$. The vertices of all graphs are 0-indexed, as dictated by the sparse6 format. We start by defining G_4^{MC} and S_4^{MC}:

```
>>sparse6<<:~?@{`oQ?bOCHBGIPBHECdOmReom\\A?aECwaAEGKCEiKIGi[NfG?
   PCwka_AEijomS_qIadPetIJ[DFaOlgqyXKbYAIqolbpi?@@qVJJmx`
   ryfhGSQDhSfKhonOhOwOYSudgKQ_OoYaPS^e@tGgH]PHhAWMC}ECCaIJS]
   eIsm@EbQ@@q@OdAgodctRbAwsfQaZNGWnOtQNGryNESzLRxwfJTmA@qasNSeDBcz@QWgQJB
   }oODDYlUDciRLCTwwVHbAXNeQWJrDfcASuWxOcNR}ELT@Q_rCrVx[eLRYZJc\\]aOh?UH{
   cOSadKcTMmsX\\YgsdQdiZNstTfBPOVwSHJcIOFspXbcHZWn

 0 2 6 7 13 15 16 17 19 20 21 24 25 27 29 31 33 35 36 37 40 42 47 49 50 54
     55 56 58 60 61 62 63 65 67 69 72 73 74 77 78 79 83 84 85 86 89 96 99
     101 102 103 104 105 110 111 112 113 114 117 120 121 123
```

G_3^{IS} and S_3^{IS} are defined by the following.

```
>>sparse6<<:cb?gGMGE_OGo]FBDoggiAGabYCETCESa\\YGHFTGC}aTLObPmcVLqQ|
   KrKNraaAIXO~\n

0 15 29 6 4 14 20 28 13 7 12 35 16 30 31 22 32
```

Finally, G_4^{IS} and S_4^{IS} are defined by the following.

```
>>sparse6<<:~?Ab_O?_g@a??@gMa_K_WJ`WNacX?cD?sMbwA`?P_O[cWF@_Q_CF@Kf?Wa`o\\
   CK[C{]Ck_dgWBS]C_f`OLD?jeG^cS@egdc_ea[x@?u_c{?sVBKXf?}bSDAwhgW@ECQF[
   B@WZE{LCOk`OWEstFs?B{QgSHE?}gCsHcDB?khkkEG|fKjaGlHSH@XBbWw__o`
   @S_wkHSRID[@?NEcReXX`OjJkvF`\\JsKAXLIS]DKta_sHHTbGqH{CGPQa?ybW|K{
   PBgeJcICPCH[``GwFHMbhBI[eHKNBcSEGr_wzJs~J@ne@JkkvK[{J[Ef`Magz`_wJ\\
   IIXvePWapCgH\\L[VEX?g@AM`tms[BwzIS@BotLs_E`RNtYMPuagbM`vc_eI|
   gMXwdpGNlbMXtb@LKxs`xVKQGgPPdPallxPINfPE_OUItZLYLQKBNAFaPJNi@dhxP{
   yNYIQK\\MAUeHOMxyaHOKqMdPFJQ@ip{OIQ`h^NaLfxmN|GLyDRCMKYMP|ZLXpQCEEQSRv
   \n

 0 1 3 4 7 8 9 10 11 12 13 15 18 24 26 27 28 31 34 40 42 43 44 46 48 53 54
     55 56 59 60 61 62 65 75 77 78 82 84 90 91 93 94 98 100 101 103 105
    107 109 110 112 114 123 124 125 127 128 129 132 134 135 136 138 141
    144 145 146 150 153 154 159 160 162
```

The following properties can easily be verified and immediately imply Theorem 2.1.

1. G_3^{IS} is a 3-regular Ramanujan graph on 36 vertices. G_4^{MC} and G_4^{IS} are 4-regular Ramanujan graphs on 124 and 163 vertices respectively.
2. S_3^{IS} and S_4^{IS} are independent sets in G_3^{IS} and G_4^{IS} respectively with $|S_3^{\mathrm{IS}}| = 17$ and $|S_4^{\mathrm{IS}}| = 74$. Consequently, $\mathrm{IS}(G_3^{\mathrm{IS}}) \geq 17/36$, and $\mathrm{IS}(G_4^{\mathrm{IS}}) \geq 74/163$.
3. The number of edges in G_4^{MC} with exactly one endpoint in S_4^{MC} is equal to 226. Consequently, $\mathrm{MC}(G_4^{\mathrm{MC}}) \geq 113/124$.

It can be easily verified that all the graphs are Ramanujan proving the claimed lower bounds on γ_4^{MC}, γ_3^{IS}, and γ_4^{IS}.

B.2 Proof of Theorem 2.2

The goal of this section is to prove improved upper bounds on σ_d^{MC} and σ_d^{IS}. The same upper bounds apply to γ_d^{MC} and γ_d^{IS}, as will be clear in the proof. We will give a detailed argument for the Max-Cut case, and sketch the (minor) changes required for Max-Independent-Set at the end.

As with previous refutation algorithms [Hof03, Hae21], our efficient certificate will be a bound on the *spectral expansion* of a random graph $G \sim \mathcal{G}(n,d)$, which is defined as $\lambda^*(G) = \max_{i>1} |\lambda_i|$ where $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ are the eigenvalues of the adjacency matrix $\mathbf{A}$ of the graph G. Friedman's theorem proves that λ^* concentrates around $2\sqrt{d-1}$.

Theorem B.1 ([Fri08]). *For all $d \in \mathbb{N}$ and $\varepsilon > 0$, with probability $1 - o_n(1)$ over $G \sim \mathcal{G}(n,d)$, we have $\lambda^*(G) \leq 2\sqrt{d-1} + \varepsilon$.*

Definition B.1 (d-ary tree). *The d-ary tree of depth L, $T_{d,L}$ is a rooted tree, where the root has d children, and all non-root vertices at distance $< L$ from the root have $d-1$ children.*

Our improvement on Hoffman's classical bounds [Hof03, Hae21] consists of concluding stronger bounds on $\mathrm{MC}(G)$ and $\mathrm{IS}(G)$ given that $\lambda^*(G) \leq \lambda$ for $\lambda = 2\sqrt{d-1} + \varepsilon$. Let us denote by $\Omega_{d,\lambda}$ the set of d-regular graphs G such that $\lambda^*(G) \leq \lambda$. We will require the notion of a *labeled* d-ary tree, which we define below.

Definition B.2 (Labeled d-ary tree). *A $\{\pm 1\}$-labeling of $T_{d,L}$ is a mapping y from the vertices of $T_{d,L}$ to $\{\pm 1\}$.*

Let $\Lambda_{d,L}$ be the collection of distributions over $\{\pm 1\}$-labelings of $T_{d,L}$, where the vertices are labeled by elements of $\{\pm 1\}$. For most of the remainder of this section, we will show how we obtain our upper bounds on $\mathrm{MC}(G)$ given $G \in \Omega_{d,\lambda}$. Our upper bound will be parametrized by a nonnegative integer $L \in \mathbb{N}$. Interestingly, we will exactly recover Hoffman's bound [Hof03] when $L = 1$. The idea is essentially to write a *finite* LP relaxation of the problem of finding the supremum of the cut fraction $|\delta_G(S, \bar{S})|/|E(G)|$ over all $G \in \Omega_{d,\lambda}$, by projecting all the information about G and S onto "local" neighborhoods at distance L.

Let $G \in \Omega_{d,\lambda}$ be an n-vertex graph, and let $S \subseteq V(G)$ be a cut in G. Let $\alpha \in [0, 1/2]$ be a value that approximates $|S|/n$, in the sense that $|\alpha - |S|/n| \leq \delta$. Define the cut vector $\mathbf{x} \in \{-1,1\}^n$ by $x_i = (-1)^{\mathbf{1}\{i \in S\}}$. We will associate the pair (G, S) with a distribution $\mu_{G,S} \in \Lambda_{d,L}$, which can be sampled from as follows:

- Sample a random ordering of the neighboring edges of each vertex in G.
- Sample a random vertex $i \sim [n]$.
- We will label $T_{d,L}$ by the values of x at all length $\leq L$ nonbacktracking walks from i (that is, walks that don't use the same edge twice in a row). Formally, a vertex v of $T_{d,L}$ can be written as a sequence $(q_1, \ldots, q_k)$ for $0 \leq k \leq L$, $1 \leq q_1 \leq d$, and $1 \leq q_l \leq d-1$ for $l > 1$. We will label v by the x value of the final vertex in the corresponding path starting from i in G. In particular, let $i_0 = i$, and for $1 \leq l \leq k$ let i_l be the q_l^{th} neighbor of i_{l-1} in the sampled ordering, excluding i_{l-2} if $l > 1$. We will label v by $y_v := x_{i_k}$.
- Output the resulting labeling y of $T_{d,L}$.

We will deduce some linear constraints on $\mu_{G,S}$, in terms of its probability distribution function.

Lemma B.1 (Local Consistency Constraints). *Let r_0 be the root of $T_{d,L}$, and let r_1 be a child of r_0. For $i \in \{0,1\}$, let T_i be the subtree of $T_{d,L}$ rooted at r_i containing all vertices at distance at most L from r_{1-i}. Then, T_0 is isomorphic to T_1, and the marginal distribution of $\mu_{G,S}$ on T_0 is identical (up to the isomorphism) of that on T_1.*

Proof. The marginal distribution on T_0 can be viewed as the following: sample a random vertex i in G and a random neighbor $j \sim i$. Output the labels x at all endpoints of a nonbacktracking walk from i of length at most L if (i,j) is the first edge in the walk, and length at most $L-1$ otherwise.

The marginal distribution on T_1 is identical, with the roles of i and j switched. Since the distribution of (i,j) is identical to that of (j,i), both marginal distributions are identical. □

Lemma B.2 (Average Value of root). *We have $2\alpha - 1 - 2\delta \leq \mathbb{E}_{y \sim \mu_{G,S}}[y_0] \leq 2\alpha - 1 + 2\delta$.*

Proof. We can directly compute $\mathbb{E}_{y \sim \mu_{G,S}}[y_0] = \mathbb{E}_{i \sim [n]}[x_i] = 2|S|/n - 1$. The sandwiching inequalities follow from the fact that $|\alpha - |S|/n| \leq \delta$. □

Lemma B.3 (Spectral Constraints). *The bound $\lambda^*(G) \leq \lambda$ implies the bounds*

$$(2\alpha - 1 - 2\delta)^2 \leq \mathbb{E}_{y \sim \mu_{G,S}, \ell}[y_0 \cdot y_\ell] \leq (2\alpha - 1 + 2\delta) + (\lambda/d)^L$$

if L is even, and

$$(2\alpha - 1 - 2\delta)^2 - (\lambda/d)^L \leq \mathbb{E}_{y \sim \mu_{G,S}, \ell}[y_0 \cdot y_\ell] \leq (2\alpha - 1 + 2\delta)^2 + (\lambda/d)^L$$

if L is odd. Here 0 denotes the root, and ℓ denotes the (random) endpoint of an L-step random walk in $T_{d,L}$ starting from the root.

Proof. We begin by computing

$$\mathbb{E}_{y\sim\mu_{G,S},\ell}[y_0 \cdot y_\ell] = \frac{\mathbf{x}^\top (\mathbf{A}/d)^L \mathbf{x}}{n} = \frac{\mathbf{x}^\top \mathbf{A}^L \mathbf{x}}{d^L \cdot n}.$$

Let us write $\mathbf{A} = d\frac{\mathbf{1}\mathbf{1}^\top}{n} + \widetilde{\mathbf{A}}$, where $d = \lambda_1$ is the trivial eigenvalue of $\mathbf{A}$ and $\widetilde{\mathbf{A}}$ is the portion of $\mathbf{A}$ orthogonal to the trivial eigenvector $\mathbf{1}$. That is, $\widetilde{A} \cdot \mathbf{1} = \mathbf{0}$. Note that $\|\widetilde{\mathbf{A}}\|_{\mathsf{op}} \leq \lambda$. Let us proceed with our computation.

$$\begin{aligned}\mathbb{E}_{y\sim\mu_{G,S},\ell}[y_0 \cdot y_\ell] &= \frac{d^L \cdot \mathbf{x}^\top \mathbf{1}\mathbf{1}^T \mathbf{x}}{d^L \cdot n^2} + \frac{\mathbf{x}^\top \widetilde{\mathbf{A}}^L \mathbf{x}}{d^L \cdot n} \\ &= \frac{(2|S| - n)^2}{n^2} + \frac{\mathbf{x}^\top \widetilde{\mathbf{A}}^L \mathbf{x}}{d^L \cdot n}.\end{aligned}$$

The Lemma follows from (1) the fact that $|\alpha - |S|/n| \leq \delta$, and (2) the bounds $|\mathbf{x}^\top \widetilde{\mathbf{A}}^L \mathbf{x}| \leq \|\widetilde{\mathbf{A}}\|_{\mathsf{op}} \cdot \|\mathbf{x}\|_2^2 \leq \lambda n$, along with the bound $\mathbf{x}^\top \widetilde{A}^L \mathbf{x} \geq 0$ if L is even. □

Lemma B.4 (Objective Value)**.** *The cut fraction $|\delta_G(S, \bar{S})|/|E(G)|$ equals $(1 - \mathbb{E}_{y\sim\mu_{G,S},i\sim[d]}[y_0 \cdot y_i])/2$, where 0 denotes the root, and i denotes the i^{th} neighbor of the root of the d-regular tree of depth L.*

Proof. This is again a direct computation. We can write

$$\mathbb{E}_{y\sim\mu_{G,S},i\sim[d]}[y_0 \cdot y_i] = \frac{\mathbf{x}^\top \mathbf{A}\mathbf{x}}{nd} = 1 - 2\frac{|\delta_G(S, \bar{S})|}{|E(G)|},$$

and rearranging gives the identity. □

Lemmas B.1 to B.3 are all linear constraints in the pdf of the distribution $\mu = \mu_{G,S}$, and Lemma B.4 is a linear objective function. We may solve this LP in the variables μ for small values of L and d to obtain valid upper bounds on objective function, the cut value $|\delta_G(S, \bar{S})|/|E(G)|$. If we solve this LP for a collection of discrete values $\alpha \in \{0, \delta, 2\delta, \ldots, 1\}$, we obtain an upper bound on the cut value of *any* cut of a graph $G \in \Omega_{d,\lambda}$.

Computational Efficiency. The above LP appears intractable – it has 2^{46} variables even for $d = 3$, $L = 4$. We apply two optimizations to make it more tractable in practice.

1. Use the same variable for isomorphic labelings of $T_{d,L}$. In particular, if there is a graph isomorphism of $T_{d,L}$ that maps a labeling y to another labeling y', then Lemma B.1 implies that they should receive the same probability in μ.
2. Only consider "locally optimal" labelings of $T_{d,L}$. To do this, we apply a preprocessing step, where we ensure that the cut S is locally optimal in the sense that for any vertex $i \in S$, the labeling of $T_{d,L}$ generated by restricting $\mathbf{x} \in \{\pm 1\}^n$ to the L-step neighborhood of i corresponds to a maximum cut of $T_{d,L}$, subject to the "boundary conditions" of labels of the leaves of $T_{d,L}$. Therefore, we can also discard any variables corresponding to such suboptimal labelings of $T_{d,L}$.

Recall that $\lambda = 2\sqrt{d-1} + \varepsilon$ for arbitrarily small ε. We will now pick $\varepsilon = 10^{-5}$. Using the above optimizations, we are able to solve the $1/\delta$ LPs for $d = 3, L = 4$ (with 4396 variables), and $d = 4, L = 2$ (with 88 variables) for $\delta = 0.0005$, resulting in the universal upper bounds $\mathrm{MC}(G) \leq 0.953$ for $G \in \Omega_{3,\lambda}$ and $\mathrm{MC}(G) \leq 0.916$ for $G \in \Omega_{4,\lambda}$.

An interesting point is that the $d = 4, L = 3$ case has 11880 variables and is technically within reach of current LP solvers, but resulted in the *same* bound on σ_4^{MC} as the $d = 4, L = 2$ case.

Upper bounds on IS(G). We conclude with some comments on the modifications needed to obtain upper bounds on σ_d^{IS}. The main modification is that we need to restrict ourselves to distributions in $\Lambda_{d,L}$ that are supported only on $\{\pm 1\}$-labelings that correspond to actual Independent Sets of $T_{d,L}$. That is, we only consider labelings y such that for any edge $\{i,j\}$ in $T_{d,L}$, y_i and y_j are not both equal to 1. The objective function disappears, and we now only need to find the largest possible value α such that the LP is still feasible. The $1/\delta$ LPs are tractable for $d = 3, L = 4$ (with 1771 variables) and $d = 4, L = 3$ (with 8855 variables), and they result in the bounds $\sigma_3^{\mathrm{IS}} \leq 0.476$ and $\sigma_4^{\mathrm{IS}} \leq 0.457$.

C Proofs of Theorems 3.1 and 3.2

In this section, we give the details of the NP-hardness of approximation proofs for MAX-k-CUT for $k \in \{3,4\}$. For ease of exposition, we begin by reintroducing all predicates needed for the proof. All predicates will be over the alphabet $\mathbb{Z}_k$.

- $\mathtt{P}_2^{\neq}$ denotes a binary predicate defined by $\mathtt{P}_2^{\neq}(x,y) = \mathbf{1}\{x \neq y\}$.
- For $i \in \{0,\ldots,k-1\}$, $\mathtt{P}_{3,k}^{\equiv i}$ denotes a 3-ary predicate defined by $\mathtt{P}_{3,k}^{\equiv i}(x,y,z) = \mathbf{1}\{x+y+z \equiv i \pmod k\}$.

Recall that MAX-k-CUT is CSP$(\mathtt{P}_2^{\neq})$, and 3LIN(k) is CSP $\left(\mathtt{P}_{3,k}^{\equiv 0},\ldots,\mathtt{P}_{3,k}^{\equiv (k-1)}\right)$. We now formally define gadgets, specifically in the context of the reduction from 3LIN(k) to MAX-k-CUT. We will need k separate gadgets. In particular, for each defining predicate $\mathtt{P}_{3,k}^{\equiv i}$ of 3LIN(k), we will need an i-gadget, which we define below.

Definition C.1 (Gadgets)**.** *Let $k \geq 2$.*

- ***i-gadget**: For every $i \in \mathbb{Z}_k$, we introduce a gadget $\mathtt{I}_{3,k}^{\equiv i}$ that reduces the predicate $\mathtt{P}_{3,k}^{\equiv i}$ to an instance of* MAX-k-CUT*. Formally, $\mathtt{I}_{3,k}^{\equiv i}$ is an instance of* MAX-k-CUT *over $3+k+n_{\mathrm{aux}}$ variables, where $n_{\mathrm{aux}} \in \mathbb{N}$.*
- **Primary, global, and auxiliary variables:** *We partition the set of variables $[3+k+n_{\mathrm{aux}}]$ into three* primary variables $\{1,2,3\}$*, k* global variables $\{4,5,\ldots,k+3\}$*, and n_{aux}* auxiliary variables $\{k+4,\ldots,3+k+n_{\mathrm{aux}}\}$*. We will always write an assignment to the variables as a tuple $(\mathbf{x},\mathbf{y},\mathbf{z}) \in \mathbb{Z}_k^{3+k+n_{\mathrm{aux}}}$ where $\mathbf{x} \in \mathbb{Z}_k^3$, $\mathbf{y} \in \mathbb{Z}_k^k$, and $\mathbf{z} \in \mathbb{Z}_k^{n_{\mathrm{aux}}}$.*

The variables $\mathbf{y}$ will be restricted in a way to break the symmetries of MAX-k-CUT. Concretely, let Y be the set of strings $\mathbf{y}$ in $\mathbb{Z}_k^k$ that are lexicographically not larger than any string $\mathbf{y}^\sigma = (\sigma(y_1),\sigma(y_2),\ldots,\sigma(y_k))$ that is obtained by permuting the alphabet $\mathbb{Z}_k$ by some permutation $\sigma : \mathbb{Z}_k \to \mathbb{Z}_k$. In particular, note that the vector $(0,1,2,\ldots,k-1)$ is in Y. We will only ever assign $\mathbf{y}$ to values in Y.

Definition C.2 (Soundness and completeness)**.** *We will set $C_i = \left(\mathtt{P}_{3,k}^{\equiv i}\right)^{-1}(1)$ and $S_i = \left(\mathtt{P}_{3,k}^{\equiv i}\right)^{-1}(0)$ to be the set of satisfying and unsatisfying assignments for the source predicate $\mathtt{P}_{3,k}^{\equiv i}$ respectively.*

We will associate four real-valued parameters to the gadget $\mathtt{I}_{3,k}^{\equiv i} = \sum_{j\in[m]} \zeta_j$ as follows.

- *(***Completeness analysis***) $c(\mathtt{I}_{3,k}^{\equiv i})$ is defined as*

$$c(\mathtt{I}_{3,k}^{\equiv i}) = \min_{\mathbf{x}\in C_i} \max_{\mathbf{z}\in\mathbb{Z}_k^{n_{\mathrm{aux}}}} \mathtt{I}_{3,k}^{\equiv i}(\mathbf{x},\mathbf{y},\mathbf{z})$$

  for $\mathbf{y} = (0,1,\ldots,k-1)$. For $\mathbf{x} \in C_i$, we say that $\arg\max_{\mathbf{z}\in\mathbb{Z}_k^{n_{\mathrm{aux}}}} \mathtt{I}_{3,k}^{\equiv i}(\mathbf{x},\mathbf{y},\mathbf{z})$ *is the* witness *for $\mathbf{x}$.*

- *(***Soundness analysis***) $s_1(\mathtt{I}_{3,k}^{\equiv i})$ is defined as*

$$s_1(\mathtt{I}_{3,k}^{\equiv i}) = \max_{\mathbf{x}\in C_i,\mathbf{z}\in\mathbb{Z}_k^{n_{\mathrm{aux}}},\mathbf{y}\in Y} \mathtt{I}_{3,k}^{\equiv i}(\mathbf{x},\mathbf{y},\mathbf{z}).$$

  Also, $s_2(\mathtt{I}_{3,k}^{\equiv i})$ is defined as

$$s_2(\mathtt{I}_{3,k}^{\equiv i}) = \max_{\mathbf{x}\in S_i,\mathbf{z}\in\mathbb{Z}_k^{n_{\mathrm{aux}}},\mathbf{y}\in Y} \mathtt{I}_{3,k}^{\equiv i}(\mathbf{x},\mathbf{y},\mathbf{z}).$$

- *(**Number of clauses**) $\mathtt{count}(\mathtt{I}_{3,k}^{\equiv i})$ is defined to be the number of clauses present in $\mathtt{I}_{3,k}^{\equiv i}$.*

We will show that any collection of i-gadgets for all $i \in \mathbb{Z}_k$ provides a reduction whose performance depends on the various parameters defined above. After this, in Appendix C.1, we give an explicit description of our gadgets and state the parameters achieved by them.

The starting point of our reduction will be the following theorem due to Håstad.

Theorem C.1 ([Hås01])**.** *Let $k \geq 2$. Given an instance $\mathtt{I} = \sum_{j\in[m]} \zeta_j$ of 3LIN(k) with m/k clauses from each of the predicates $\mathtt{P}_{3,k}^{\equiv 0}, \mathtt{P}_{3,k}^{\equiv 1}, \ldots, \mathtt{P}_{3,k}^{\equiv (k-1)}$, it is NP-hard to distinguish between the following cases for any $\varepsilon > 0$:*

1. **Completeness case**. $\max_{\mathbf{x}} \mathtt{I}(\mathbf{x}) \geq (1-\varepsilon)\cdot m$, *or*
2. **Soundness case**. $\max_{\mathbf{x}} \mathtt{I}(\mathbf{x}) \leq (1/k+\varepsilon)\cdot m$.

Theorem C.2 details how our gadgets imply NP-hardness of approximation for MAX-k-CUT, with $k \in \{3,4\}$.

Theorem C.2. *Let $k \geq 2$. Let $\mathtt{I}_{3,k}^{\equiv i}$ be an i-gadget that reduces the predicate $\mathtt{P}_{3,k}^{\equiv i}$ to an instance of* MAX-k-CUT *for each $i \in \mathbb{Z}_k$. For any $\varepsilon > 0$, given an instance $\mathtt{J}$ of* MAX-k-CUT *with M constraints, it is NP-hard to distinguish between the following two cases for any $\varepsilon > 0$:*

1. **Completeness case***: $\max_{\mathbf{x}} \mathtt{J}(\mathbf{x}) \geq (a-\varepsilon)\cdot M$, or*
2. **Soundness case***: $\max_{\mathbf{x}} \mathtt{J}(\mathbf{x}) \leq (b+\varepsilon)\cdot M$,*

where

$$a = \frac{\sum_{i\in\mathbb{Z}_k} c(\mathtt{I}_{3,k}^{\equiv i})}{\sum_{i\in\mathbb{Z}_k} \mathtt{count}(\mathtt{I}_{3,k}^{\equiv i})}, \quad b = \frac{s_1(\mathtt{I}_{3,k}^{\equiv r_0}) + \sum_{i\in\mathbb{Z}_k\setminus\{0\}} s_2(\mathtt{I}_{3,k}^{\equiv r_i})}{\sum_{i\in\mathbb{Z}_k} \mathtt{count}(\mathtt{I}_{3,k}^{\equiv i})}.$$

Here $r_0, r_1, \ldots r_{k-1}$ is a permutation of $\{0,1,\ldots,k-1\}$ such that $s_1(\mathtt{I}_{3,k}^{\equiv r_i}) - s_2(\mathtt{I}_{3,k}^{\equiv r_i})$ is maximized by r_0.

Proof of Theorem C.2. We will perform a reduction starting from the NP-hard problem in *Theorem* C.1 to MAX-k-CUT. Let $\mathtt{I} = \sum_{j\in[m]} \zeta_j$ be an n-variable instance of 3LIN(k), and suppose that it has m/k clauses using $\mathtt{P}_{3,k}^{\equiv i}$ for all $i \in \mathbb{Z}_k$. We will show how to reduce $\mathtt{I}$ to an instance $\mathtt{J}$ of MAX-k-CUT.

Let n_{aux} be the number of auxiliary variables in all gadgets $\{\mathtt{I}_{3,k}^{\equiv i} : i \in \mathbb{Z}_k\}$; we can assume they all have the same number of auxiliary variables by introducing new dummy variables to some of them. J will be defined as an instance on $n + k + n_{\text{aux}}\cdot m$ variables, which we will denote as $\mathbf{x} \in \mathbb{Z}_k^n$, $\mathbf{y} \in \mathbb{Z}_k^k$, and $\mathbf{z} \in \mathbb{Z}_k^{n_{\text{aux}}\cdot m}$. For $j \in [m]$ we will denote the j^{th} block of size n_{aux} in z as z^j. We will refer to $\mathbf{x}, \mathbf{y}, \mathbf{z}$ as the primary, global, and auxiliary variables respectively.

Let $\zeta_j = \mathtt{P}_{3,k}^{\equiv i}(x_{j_1}, x_{j_2}, x_{j_3})$ be a clause corresponding to the predicate $\mathtt{P}_{3,k}^{\equiv i}$ for some i. We will add a copy of $\mathtt{I}_{3,k}^{\equiv i}$ on the variables $(\mathbf{x}', \mathbf{y}, \mathbf{z}^j)$, where $\mathbf{x}' = (x_{j_1}, x_{j_2}, x_{j_3})$. Note that the number of clauses in $\mathtt{J}$ is $M = (m/k)\cdot\sum_{i\in\mathbb{Z}_k} \mathtt{count}(\mathtt{I}_{3,k}^{\equiv i})$.

Now we will invoke Theorem C.1, which says that it is NP-hard to distinguish between the cases that $\mathtt{I}$ is almost completely satisfiable, and $\mathtt{I}$ is essentially $1/k$-satisfiable.

1. **Completeness case**: Assuming there is an $\mathbf{x} \in \mathbb{Z}_k^n$ such that $\mathtt{I}(\mathbf{x}) \geq (1-\varepsilon)\cdot m$, we show how to construct an assignment $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathbb{Z}_k^{n+k+n_{\text{aux}}\cdot m}$ such that $\mathtt{J}(\mathbf{x}, \mathbf{y}, \mathbf{z}) \geq a\cdot M$. We will set $\mathbf{y} = (0, 1, 2, \ldots, k-1)$, and for each clause $\zeta_j = \mathtt{P}_{3,k}^{\equiv i}(x_{j_1}, x_{j_2}, x_{j_3})$, $\mathbf{z}^j$ will be set as the witness
   $\mathbf{z}^j = \arg\max_{\mathbf{z}\in\mathbb{Z}_k^{n_{\text{aux}}}} \mathtt{I}_{3,k}^{\equiv i}(\mathbf{x}', \mathbf{y}, \mathbf{z})$, where $\mathbf{x}' = (x_{j_1}, x_{j_2}, x_{j_3})$.
   It is immediate that $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ satisfies at least

$$\sum_{i\in\mathbb{Z}_k} c(\mathtt{I}_{3,k}^{\equiv i})\cdot m/k - O(\varepsilon\cdot m) = a\cdot M - O(\varepsilon)\cdot m \geq (a-\varepsilon')\cdot M$$

   clauses of J, where $\varepsilon' = O(\varepsilon)$ can be chosen to be arbitrarily close to 0.

2. **Soundness case**: We must show that if $\max_{\mathbf{x}} \mathtt{I}(\mathbf{x}) \leq (1/k + \varepsilon) \cdot m$, then $\max_{\mathbf{x},\mathbf{y},\mathbf{z}} \mathtt{J}(\mathbf{x},\mathbf{y},\mathbf{z}) \leq (b + \varepsilon') \cdot M$. Let $(\mathbf{x},\mathbf{y},\mathbf{z})$ be an assignment to the variables of $\mathtt{J}$.
The first step we perform is to assume without loss of generality that $\mathbf{y} \in Y$. To do this, note that by definition of Y, there is some permutation $\sigma : \mathbb{Z}_k \to \mathbb{Z}_k$ such that $\mathbf{y}^\sigma = (\sigma(y_1), \ldots, \sigma(y_k)) \in Y$. We will apply σ to all variables in the full assignment $(\mathbf{x},\mathbf{y},\mathbf{z})$, resulting in no change in the value of $\mathtt{J}(\mathbf{x},\mathbf{y},\mathbf{z})$ while guaranteeing that $\mathbf{y} \in Y$.
For $i \in \mathbb{Z}_k$, let α_i denote the fraction of $\mathtt{P}^{\equiv i}_{3,k}$ constraints of I that are satisfied by $\mathbf{x}$. By assumption, we have $\sum_{i \in \mathbb{Z}_k} \alpha_i \leq 1 + k\varepsilon$. Let us define $\hat{\alpha}_i = \alpha_i / (\sum_{i' \in \mathbb{Z}_k} \alpha_{i'})$, so $\sum_{i \in \mathbb{Z}_k} \hat{\alpha}_i = 1$. In other words, $\{\hat{\alpha}_i : i \in \mathbb{Z}_k\}$ defines a probability distribution.
On the other hand, since $y \in Y$ we can now bound $\mathtt{J}(\mathbf{x},\mathbf{y},\mathbf{z})$ as follows.

$$\begin{aligned}
\mathtt{J}(\mathbf{x},\mathbf{y},\mathbf{z}) &\leq m \cdot \left(\sum_{i \in \mathbb{Z}_k} \alpha_i \cdot s_1(\mathtt{I}^{\equiv i}_{3,k}) + (1 - \alpha_i) \cdot s_2(\mathtt{I}^{\equiv i}_{3,k}) \right) \\
&\leq m \cdot \left(\sum_{i \in \mathbb{Z}_k} \hat{\alpha}_i \cdot s_1(\mathtt{I}^{\equiv i}_{3,k}) + (1 - \hat{\alpha}_i) \cdot s_2(\mathtt{I}^{\equiv i}_{3,k}) \right) + O(\varepsilon) \cdot m \qquad (\textstyle\sum_{i \in \mathbb{Z}_k} \alpha_i \leq 1 + k\varepsilon) \\
&= m \cdot \left(\sum_{i \in \mathbb{Z}_k} \hat{\alpha}_i \cdot (s_1(\mathtt{I}^{\equiv i}_{3,k}) - s_2(\mathtt{I}^{\equiv i}_{3,k})) + s_2(\mathtt{I}^{\equiv i}_{3,k}) \right) + O(\varepsilon) \cdot m.
\end{aligned}$$

Note that subject to $\hat{\alpha}_i$ being a probability distribution, this is maximized when $\hat{\alpha}_i$ is a point mass on a maximizer of $s_1(\mathtt{I}^{\equiv i}_{3,k}) - s_2(\mathtt{I}^{\equiv i}_{3,k})$. Therefore, this is bounded by

$$m \cdot \left(s_1(\mathtt{I}^{\equiv r_0}_{+,}) \sum_{i \in \mathbb{Z}_k \setminus \{0\}} s_2(\mathtt{I}^{\equiv r_i}_{3,k}) \right) + O(\varepsilon) \cdot m = (b + O(\varepsilon)) \cdot M = (b + \varepsilon') \cdot M,$$

where ε' can be chosen to be arbitrarily close to 0. This completes the proof of the soundness case.

□

C.1 Description of our gadgets

For $k \in \{3,4\}$, we used AlphaEvolve to find gadgets $\{\mathtt{I}^{\equiv i}_{3,k} : i \in \mathbb{Z}_k\}$, scoring a particular gadget by the inverse inapproximability ratio a/b, where a and b are defined in Theorem C.2. We now concretely define the specific gadgets found by AlphaEvolve and list the parameters they achieve (Definition C.2). We derive the final inapproximability results (Theorems 3.1 and 3.2) by applying Theorem C.2 on these gadgets.

We have formatted a gadget as a weighted "edge list" corresponding to a collection of $\mathtt{P}^{\neq}_2$ clauses. The format is meant to be understood as follows: a tuple (a, b, w) corresponds to w parallel edges from variable a to variable b, that is, the gadget contains w copies of the term $\mathtt{P}^{\neq}_2$ applied to variables a and b.

Gadgets for MAX-3-CUT. We write the list of edges for $\mathtt{I}^{\equiv 0}_{3,3}$ first, which has 12 total variables, and hence 6 auxiliary variables. Note that the global variables $\{4,5,6\}$ are not used in this gadget. See Figure 2a for a visual representation of the gadget.

```
[(1, 7, 1), (1, 8, 1), (1, 9, 1), (1, 10, 1), (2, 9, 1), (2, 10, 1), (2, 11,
   1), (2, 12, 1), (3, 7, 1), (3, 8, 1), (3, 11, 1), (3, 12, 1), (7, 12, 1),
   (7, 10, 1), (8, 9, 1), (8, 11, 1), (9, 12, 1), (10, 11, 1)]
```

Below (and in Figure 2b), we write the list of edges for our $\mathtt{I}^{\equiv 1}_{3,3}$ gadget, which has $6 + n_{\text{aux}} = 14$ total variables, and hence 8 auxiliary variables. Note that repeated edges correspond to multiple $\mathtt{P}^{\neq}_2$ clauses on the same pair of variables.

```
[(3, 7, 1), (4, 6, 3), (4, 12, 1), (3, 13, 2), (5, 10, 1), (9, 11, 1), (11,
   14, 1), (1, 3, 2), (1, 9, 1), (2, 8, 1), (2, 14, 1), (13, 14, 2), (6, 11,
   1), (4, 5, 3), (3, 9, 1), (5, 6, 3), (4, 8, 2), (5, 9, 2), (1, 2, 2), (2,
   7, 1), (10, 14, 1), (1, 8, 1), (1, 14, 1), (2, 13, 2), (6, 7, 2), (7, 12,
   1), (4, 7, 1), (12, 14, 1), (3, 8, 1), (3, 14, 1), (5, 8, 1), (8, 10, 1),
   (2, 3, 2), (2, 9, 1), (1, 7, 1), (1, 13, 2), (6, 9, 1)]
```

$\mathtt{I}_{3,3}^{\equiv 2}$ is defined identically to $\mathtt{I}_{3,3}^{\equiv 1}$, except with the global variables reordered from $(4,5,6)$ to $(4,6,5)$. We can calculate the relevant parameters of these gadgets as below.

Lemma C.1. *Let $I_{3,3}^{\equiv 0}$, $I_{3,3}^{\equiv 1}$, and $I_{3,3}^{\equiv 2}$ be as above. $I_{3,3}^{\equiv 0}$ is a 0-gadget that maps the predicate $P_{3,3}^{\equiv 0}$ to an instance of* MAX-3-CUT*, with parameters $c(I_{3,3}^{\equiv 0}) = s_1(I_{3,3}^{\equiv 0}) = t(I_{3,3}^{\equiv 0}) = 18$, and $s_2(I_{3,3}^{\equiv 0}) = 16$. For $i \in \{1,2\}$, $I_{3,3}^{\equiv i}$ is an i-gadget with parameters $c(I_{3,3}^{\equiv i}) = s_1(I_{3,3}^{\equiv i}) = 48$, $s_2(I_{3,3}^{\equiv i}) = 46$, and $\mathtt{count}(I_{3,3}^{\equiv i}) = 53$.*

Combined with Theorem C.2, we immediately get Theorem 3.1.

Gadgets for MAX-4-CUT. Below we write the weighted edge list for our $\mathtt{I}_{3,4}^{\equiv 0}$ gadget for MAX-4-CUT.

```
[(1, 2, 866), (1, 3, 865), (1, 5, 324), (1, 6, 168), (1, 7, 324), (1, 8, 178),
    (1, 9, 1361), (1, 10, 648), (1, 11, 628), (1, 12, 731), (1, 13, 36), (1,
   14, 331), (1, 15, 1038), (1, 16, 473), (1, 17, 73), (1, 18, 442), (1, 19,
   1013), (2, 3, 866), (2, 5, 323), (2, 6, 168), (2, 7, 323), (2, 8, 1243),
   (2, 9, 1361), (2, 10, 724), (2, 11, 261), (2, 12, 731), (2, 13, 36), (2,
   14, 331), (2, 15, 65), (2, 16, 218), (2, 17, 601), (2, 18, 463), (2, 19,
   995), (3, 5, 324), (3, 6, 168), (3, 7, 322), (3, 8, 1125), (3, 9, 1360),
   (3, 10, 288), (3, 11, 719), (3, 12, 731), (3, 13, 37), (3, 14, 331), (3,
   15, 1037), (3, 16, 509), (3, 17, 603), (3, 18, 149), (3, 19, 66), (4, 5,
   1261), (4, 6, 1089), (4, 7, 1259), (4, 9, 173), (4, 10, 19), (4, 11, 11),
   (4, 12, 167), (4, 13, 1255), (4, 14, 12), (4, 16, 11), (4, 17, 9), (4, 18,
    29), (5, 6, 660), (5, 7, 1429), (5, 8, 87), (5, 9, 314), (5, 10, 34), (5,
    11, 10), (5, 12, 581), (5, 13, 2), (5, 14, 389), (5, 15, 8), (5, 16, 15),
    (5, 17, 24), (5, 18, 7), (5, 19, 16), (6, 7, 656), (6, 8, 380), (6, 10,
   181), (6, 11, 159), (6, 13, 49), (6, 15, 350), (6, 16, 78), (6, 17, 196),
   (6, 18, 79), (6, 19, 322), (7, 8, 94), (7, 9, 319), (7, 10, 35), (7, 11,
   20), (7, 12, 655), (7, 13, 1), (7, 14, 316), (7, 15, 16), (7, 16, 2), (7,
   17, 23), (7, 18, 7), (7, 19, 22), (8, 9, 52), (8, 10, 338), (8, 11, 460),
   (8, 12, 508), (8, 13, 20), (8, 14, 338), (8, 16, 326), (8, 17, 59), (8,
   18, 388), (9, 10, 361), (9, 11, 343), (9, 12, 263), (9, 13, 1265), (9, 14,
    60), (9, 15, 25), (9, 16, 288), (9, 17, 199), (9, 18, 207), (9, 19, 25),
   (10, 11, 3), (10, 12, 344), (10, 13, 226), (10, 14, 97), (10, 15, 404),
   (10, 16, 3), (10, 17, 3), (10, 18, 103), (10, 19, 82), (11, 12, 296), (11,
    13, 360), (11, 14, 160), (11, 15, 60), (11, 16, 109), (11, 17, 2), (11,
   19, 436), (12, 13, 898), (12, 15, 517), (12, 16, 120), (12, 17, 264), (12,
    18, 238), (12, 19, 448), (13, 14, 344), (13, 15, 34), (13, 16, 262), (13,
    17, 300), (13, 18, 178), (13, 19, 6), (14, 15, 190), (14, 16, 218), (14,
   17, 134), (14, 18, 66), (14, 19, 242), (15, 16, 97), (15, 17, 328), (15,
   18, 352), (16, 17, 2), (16, 19, 252), (17, 19, 357), (18, 19, 73)]
```

For $i \geq 1$, our $\mathtt{I}_{3,4}^{\equiv i}$ gadget is obtained from our $\mathtt{I}_{3,4}^{\equiv (i-1)}$ gadget by reordering the global variables from $(4,5,6,7)$ to $(7,4,5,6)$. The below Lemma can be verified computationally using Theorem C.1.

Lemma C.2. *Let $I_{3,4}^{\equiv 0}$, $I_{3,4}^{\equiv 1}$, $I_{3,4}^{\equiv 2}$, and $I_{3,4}^{\equiv 3}$ be as above. For each i, $I_{3,4}^{\equiv i}$ is a i-gadget that maps the predicate $P_{3,4}^{\equiv i}$ to an instance of* MAX-4-CUT*, with parameters $c(I_{3,4}^{\equiv i}) = 49535$, $s_1(I_{3,4}^{\equiv i}) = 49538$, $s_2(I_{3,4}^{\equiv i}) = 48681$, and $\mathtt{count}(I_{3,4}^{\equiv i}) = 52941$.*

Together with Theorem C.2, we immediately get Theorem 3.2.

Finally, for the convenience of the reader, we provide a code snippet that computes the set Y (from Definition C.1) of allowed assignments to the global variables as a function of k below.

```
# Outputs list of allowed assignments to global variables for the 3lin(k) to max-k-cut reduction
def allowed_global_assignments(k):
  # iterate over all k-tuples of elements of Z_k
  ret = []
  for y in itertools.product(range(k), repeat=k):
    # To determine whether y is lexicographically minimal with respect to permuting each entry by
        the same permutation of Z_k, we check that ith distinct element of y is equal to i-1.
    elements = []
    for element in y:
      if element not in elements:
        elements.append(element)
    if elements == list(range(len(elements))):
      ret.append(y)
  return ret
```

D Proof of the Hardness of Approximation for TSP

In this section we prove Theorem 4.1. We will begin by recalling the various predicates we consider in this section, all of which are over the boolean alphabet $\mathbb{Z}_2 = \{0, 1\}$. Following the convention in the rest of the paper, we define the predicates relevant to this section.

- $\mathrm{P}_2^{=}$ denotes a binary predicate defined by $\mathrm{P}_2^{=}(x, y) = \mathbf{1}\{x = y\}$.
- $\mathrm{P}_2^{\neq}$ denotes a binary predicate defined by $\mathrm{P}_2^{\neq}(x, y) = \mathbf{1}\{x \neq y\}$.
- $\mathrm{P}_{3,2}^{\equiv 1}$ denotes a 3-ary predicate defined by $\mathrm{P}_{3,2}^{\equiv 1}(x, y, z) = \mathbf{1}\{x + y + z \equiv 1 \pmod 2\}$.

The main technical result of this section is Theorem D.2, which is our generalized hardness statement that is formulated in terms of the soundness and completeness parameters $s(H)$ and $c(H)$ of an equation gadget H (see Definitions D.3 and D.4). Later on in Appendix D.2, we will show how to invoke Theorem D.2 with our equation gadget which achieves parameters $s(H) = c(H) = 10$ to obtain Theorem 4.1.

As we mentioned in Section 4, our reductions will be described in terms of MWST, which is an equivalent problem to metric TSP. In fact, instead of working directly with MWST, it will be more convenient to work with a variation where certain edges are "forced" to be in the spanning tours we consider.

Definition D.1 (MWST$_f$)**.** *An instance of* MWST$_f$ *is defined by a weighted graph $G = (V, E_u \cup E_f, w)$, where the edge set is split into a set E_u of "unforced" edges, and E_f of "forced" edges.*

Recall that a spanning tour is a closed walk on a graph that visits every vertex at least once. The objective of MWST$_f$ is: given an instance G, to find the minimum weight spanning tour in G that uses each forced edge at least once (and without loss of generality, one that uses each edge at most twice). With this in mind, it will be convenient to make the following definition.

Definition D.2 (Valid spanning tour)**.** *Let T be a spanning tour in a graph $G = (V, E_u \cup E_f, w)$. We say that T is a* valid *tour on G if it (1) it uses each forced edge in E_f at* least once*, and (2) it uses each edge in G at most twice.*

A simple reduction shows that approximating MWST$_f$ is equivalent to approximating MWST.

Lemma D.1 (e.g. [KLS15])**.** *Let $\varepsilon > 0$. There is a polynomial time reduction mapping an instance $G = (V, E_u \cup E_f, w)$ of* MWST$_f$ *to an instance $G' = (V, E', w')$ of* MWST*, so the following inequalities hold, where T is the minimum weight valid spanning tour in G and T' is the minimum weight spanning tour in G'.*

$$w(T) - \varepsilon \cdot \max_{e \in E_f} w(e) \leq w'(T') \leq w(T).$$

We will henceforth focus on proving inapproximability for MWST_f by reduction from 3LIN(2). As in Section 4, our reduction is defined by a gadget that determines how we map $\mathtt{P}^{\equiv 1}_{3,2}$ clauses to edges in a MWST_f instance, which we refer to as an *equation gadget*.

Definition D.3 (Equation gadget)**.** *An equation gadget (e.g., Figure 7a) is a weighted multigraph $H = (V, E_u \cup E_f, w)$ on vertex set $V = [3 + 1 + n_{\text{aux}}]$ with the following properties:*

1. ***Contact, central, and auxiliary vertices**: The $[3 + 1 + n_{\text{aux}}]$ of vertices are partitioned into* three *contact vertices $\{1, 2, 3\}$, one central vertex 4, and n_{aux} auxiliary vertices $\{5, \ldots, 3 + 1 + n_{\text{aux}}\}$.*
2. ***Unforced, forced, and special edges**: E_u and E_f are referred to as the* unforced *and* forced *edges respectively, therefore H forms an instance of MWST_f. There is a fixed set $E_s \subset E_u$ of edges referred to as* special *edges, consisting of edges $(\ell, 4)$ with weight $w((\ell, 4)) = 1/2$ for each contact vertex $\ell \in \{1, 2, 3\}$.*

Given an equation gadget H and a valid spanning tour Q in H, we define

- $k_i^H(Q)$ to be the number of special edges that appear exactly $\mathtt{I}$ times in Q. Note for example that $\sum_i k_i^H(Q) = 3$ for all valid Q.

H will be used to map the "equals 1" clause $\mathtt{P}^{\equiv 1}_{3,2}(z_1, z_2, z_3) = \mathbf{1}\{z_1 + z_2 + z_3 \equiv 1 \pmod 2\}$ of 3LIN(2) (recall that the variables z_ℓ are over $\{0, 1\}$). As intuition, we will aim to encode an assignment $\mathbf{z} \in \{0, 1\}^3$ to the variables of a $\mathtt{P}^{\equiv 1}_{3,2}$ clause as a spanning tour T such that the special edge $(i, 4)$ is used exactly $2 \cdot z_i$ times. We will require that satisfying assignments to $\mathtt{P}^{\equiv 1}_{3,2}$ admit such spanning tours (which must have $k_1(T) = 0$ and $k_2(T)$ odd) with low weight, and that spanning tours not of this form must be more expensive. We formally encapsulate this intuition in the following definition.

Definition D.4 (Soundness and Completeness)**.** *Here we provide the completeness and soundness definitions for any equation gadget $H = (V, E_u \cup E_f, w)$.*

- ***Completeness**: Let $\mathbf{z} \in \{0, 1\}^3$, and let $C = (P^{\equiv 1}_{3,2})^{-1}(1) \subset \{0, 1\}^3$ be the set of satisfying assignments of $P^{\equiv 1}_{3,2}$. We use $\mathbf{Q}_\mathbf{z}$ to refer to the set of valid spanning tours Q in H such that the special edge $(\ell, 4)$ is unused for all ℓ such that $\mathbf{z}_\ell = 0$, and is used twice otherwise. $c(H)$ is defined as*

$$c(H) = \max_{z \in C} \min_{Q \in \mathbf{Q}_\mathbf{z}} w(Q)$$

- ***Soundness**: $s(H)$ is defined as*

$$s(H) = \min_Q w(Q) - k_1^H(Q)/2 - \mathbf{1}\{k_1^H(Q) = 0\} \cdot \mathbf{1}\{k_2^H(Q) \text{ is even}\},$$

where the minimum is taken over all valid spanning tours Q.

Similarly to [CC20], we will perform a reduction from a particular "hybrid" bounded-occurrence CSP over $\{0, 1\}$ (described in Theorem D.1). Informally, the hybrid CSP is a particularly structured instance that contains a small number of $\mathtt{P}^{\equiv 1}_{3,2}$ clauses along with some simpler two-variable equality and inequality clauses, where $\mathtt{P}^{\equiv 1}_{3,2}(z_1, z_2, z_3) = \mathbf{1}\{z_1 + z_2 + z_3 \equiv 1 \pmod 2\}$. Consider the 2-ary predicates $\mathtt{P}^{=}_2(x_1, x_2) = \mathbf{1}\{x_1 = x_2\}$, $\mathtt{P}^{\neq}_2(x_1, x_2) = \mathbf{1}\{x_1 \neq x_2\}$. In Definition D.5 we provide the CSP instance we reduce from, and in Theorem D.1 we state its soundness and completeness guarantees.

Definition D.5 (Hybrid-3LIN(2) [CC20])**.** *Let k be a fixed integer. An instance of Hybrid-*3LIN(2) *takes the form $\mathtt{I} = \mathtt{I}^{=}_2 + 2 \cdot \mathtt{I}^{\neq}_2 + 2 \cdot \mathtt{I}^{\equiv 1}_{3,2}$ of $\text{CSP}(P^{\equiv 1}_{3,2}, P^{=}_2, P^{\neq}_2)$ where:*

- ***Variables**: The variables in $\mathtt{I}$ are labeled as $x_{i,j,b}$ for $i \in [n]$, $j \in [11k]$, and $b \in \{0, 1\}$. That is, there are $22kn$ variables. Each variable will participate in exactly two distinct $P^{=}_2$ clauses in $\mathtt{I}^{=}_2$. Let $S = \{11\ell : \ell \in [k]\}$ be the set of multiples of 11 in $[11k]$. Following the notation in [KLS15], we will classify a variable $x_{i,j,b}$ as a "contact" if $j \in S$, and a "checker" otherwise. A contact will additionally appear in exactly one $P^{\neq}_2$ clause in $\mathtt{I}^{\neq}_2$, and a checker will appear in exactly one $P^{\equiv 1}_{3,2}$ clause in $\mathtt{I}^{\equiv 1}_{3,2}$.*

- ***Equality clauses***: *$\mathtt{I}_2^{=}$ consists of the following: each variable $x_{i,j,b}$ participates in two clauses $\mathtt{P}_2^{=}(x_{i,j,b}, x_{i,j+1,b})$ and $\mathtt{P}_2^{=}(x_{i,j,b}, x_{i,j-1,b})$, where addition/subtraction to j is done modulo $11k$.*
- ***Inequality clauses***: *$\mathtt{I}_2^{\neq}$ is defined by the following: for each $i \in [n]$, $j \in [11k] \setminus S$, we add the clause $\mathtt{P}_2^{\neq}(x_{i,j,0}, x_{i,\pi(j),1})$, where π is a fixed bijection on $[11k] \setminus S$.*
- ***3LIN(2) clauses:*** *$\mathtt{I}_{3,2}^{\equiv 1}$ consists of a collection of $\mathtt{P}_{3,2}^{\equiv 1}$ clauses involving only contact variables, and every contact variable participates in exactly one such clause.*

Note that in the definition above, $\mathtt{I}$ contains one copy of each clause from $\mathtt{I}_2^{=}$, and two copies of each clause from $\mathtt{I}_2^{\neq}$ and $\mathtt{I}_{3,2}^{\equiv 1}$.

Theorem D.1 (Soundness and completeness for weighted-3LIN(2) [CC20])**.** *Let $\varepsilon \in (0, 1/4)$, and let k be a large enough integer depending on ε, and let $\mathtt{I}$ be an instance of the weighted-*3LIN(2) *instance from Definition D.5. Then, it is NP-hard to decide between the two following cases:*

1. ***Completeness:*** *There is an assignment $\mathbf{x} = (x_{i,j,b})_{i,j,b}$ satisfying all clauses in $\mathtt{I}_2^{=}$ and $\mathtt{I}_2^{\neq}$, and all but εnk clauses in $\mathtt{I}_{3,2}^{\equiv 1}$, or*
2. ***Soundness:*** *For all assignments $\mathbf{x} = (x_{i,j,b})_{i,j,b}$, at least $(2/3 - \varepsilon)nk$ clauses of $\mathtt{I} = \mathtt{I}_2^{=} + 2\mathtt{I}_2^{\neq} + 2\mathtt{I}_{3,2}^{\equiv 1}$ are unsatisfied.*

We prove the following result, which can be interpreted as a modularized version of the proofs in [KLS15, CC20].

Theorem D.2. *Let H be an equation gadget (Definition D.3) such that $c(H) < \infty$. For any $\varepsilon > 0$, it is NP-hard to approximate* MWST$_f$ *within $\frac{91+2\cdot s(H)}{90+2\cdot c(H)} - \varepsilon$.*

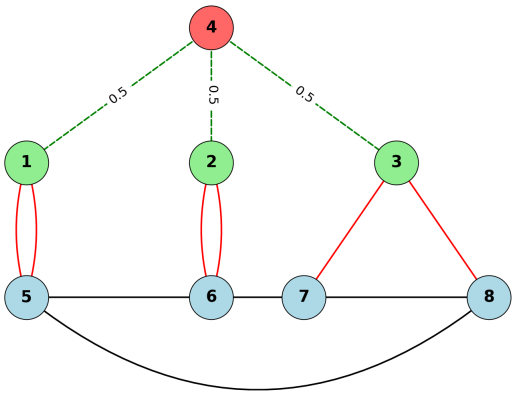
Proof. We will provide a reduction from the NP-hard problem described in Theorem D.1 to MWST$_f$. Suppose we are given a Hybrid-3LIN(2) instance $\mathtt{I} = \mathtt{I}_2^{=} + 2 \cdot \mathtt{I}_2^{\neq} + 2 \cdot \mathtt{I}_{3,2}^{\equiv 1}$ on variables $x_{i,j,b}$ for $i \in [n], j \in [11k], b \in \{0,1\}$. Our instance G of MWST$_f$ will contain a vertex for each variable, which we will also denote by $x_{i,j,b}$. Additionally we will add a single "central" vertex s. We will convert every clause of $\mathtt{I}$ into a corresponding structure in G, which we define below. We also provide a visual representation of this correspondence in Figure 7a and Figure 7b.

- **Equality clauses**: Each clause in $\mathtt{I}_2^{=}$ will be replaced by a single unforced edge in G of weight one. (In Figure 7b these edges are denoted in black.)
- **Inequality clauses**: Each clause in $\mathtt{I}_2^{\neq}$ will be replaced by two parallel forced edges in G of weight one. (In Figure 7b these edges are denoted in red.)
- **3LIN(2) clauses**: A clause $\mathtt{P}_{3,2}^{\equiv 1}(x_{i_1,j_1,b_1}, x_{i_2,j_2,b_2}, x_{i_3,j_3,b_3})$ in $\mathtt{I}_{3,2}^{\equiv 1}$ will be replaced by a copy of the gadget H (shown in Figure 7a). More precisely, for each such clause we will add n_{aux} *new* vertices, say $a_1, \ldots, a_{n_{\text{aux}}}$, and add a single induced copy of the forced and unforced edges from H on the contact vertices $x_{i_1,j_1,b_1}, x_{i_2,j_2,b_2}, x_{i_3,j_3,b_3}$, auxiliary vertices $a_1, \ldots, a_{n_{\text{aux}}}$, and central vertex s. Note that the central vertex s is shared among all $\mathtt{P}_{3,2}^{\equiv 1}$ clauses, but the auxiliary vertices are individually created for each $\mathtt{P}_{3,2}^{\equiv 1}$ clause.

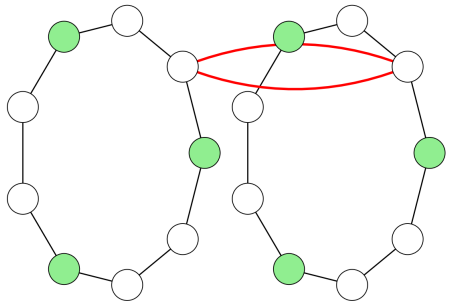
We will define $\mathtt{E}_2^{=}$, $\mathtt{E}_2^{\neq}$, and $\mathtt{E}_{3,2}^{\equiv 1}$ as the edges added in the three steps above respectively. This completes the definition of G. It remains to argue that the MWST$_f$ value of G has the desired gap in the completeness and soundness cases.

Lemma D.2 (Completeness)**.** *Suppose there is an assignment $\mathbf{x} = (x_{i,j,b})_{i,j,b}$ to the Hybrid-*3LIN(2) *instance $\mathtt{I}$ that satisfies all clauses in $\mathtt{I}_2^{=}$ and $\mathtt{I}_2^{\neq}$ clauses, and violates at most Δ clauses in $\mathtt{I}_{3,2}^{\equiv 1}$. Then, there is a valid spanning tour in G of total weight at most $nk \cdot (30 + 2 \cdot c(H)/3) + 2(\Delta + 2n) \cdot w(H)$.*

Lemma D.3 (Soundness)**.** *Suppose there is a valid spanning tour T of the* MWST$_f$ *instance G with cost at most $nk \cdot (30 + 2 \cdot s(H)/3) + \Delta$. Then there is an assignment $\mathbf{x} = (x_{i,j,b})_{i,j,b}$ that leaves at most 2Δ clauses in $\mathtt{I}$ unsatisfied.*



(a) The equation gadget discovered by AlphaEvolve. This figure is a reproduction of Figure 4.



(b) Graph encoding $\mathtt{I}_2^{=}$ and $\mathtt{I}_2^{\neq}$ clauses. The green vertices correspond to the contact vertices, and the white ones correspond to the checker vertices. The black edges encode the $\mathtt{I}_2^{=}$ clauses, and the red edges encode the $\mathtt{I}_2^{\neq}$ clauses. All edges have weight one.

Figure 7: Equation gadget discovered by AlphaEvolve, and the graph encoding of the various clauses.

Before proving Lemmas D.2 and D.3, we will use them to complete the proof of Theorem D.2. Theorem D.1 combined with Lemmas D.2 and D.3 implies that it is NP-hard to distinguish between instances G that admit a spanning tour of weight at most $nk \cdot (30 + 2 \cdot c(H)/3) + 2(\varepsilon nk + 2n) \cdot w(H)$ from instances where all spanning tours have weight at least $nk \cdot (30 + 2 \cdot s(H)/3) + nk \cdot (1/3 - \varepsilon/2)$. That is, it is NP-hard to approximate MWST_f within

$$\frac{nk \cdot (30 + 2 \cdot s(H)/3) + nk \cdot (1/3 - \varepsilon/2)}{nk \cdot (30 + 2 \cdot c(H)/3) + 2(\varepsilon nk + 2n) \cdot w(H)}.$$

Setting ε to be sufficiently small and k to be sufficiently large, this is at least

$$\frac{91 + 2 \cdot s(H)}{90 + 2 \cdot c(H)} - \varepsilon'$$

for arbitrarily small ε'. It remains to prove Lemmas D.2 and D.3, which we do in the following section. □

D.1 Proofs of Lemmas D.2 and D.3

Proof of Lemma D.2. We will construct collections of edges $\mathtt{T}_2^{=}$, $\mathtt{T}_2^{\neq}$, and $\mathtt{T}_{3,2}^{\equiv 1}$ from $\mathtt{E}_2^{=}$, $\mathtt{E}_2^{\neq}$, $\mathtt{E}_{3,2}^{\equiv 1}$, and set $T = \mathtt{T}_2^{=} \cup \mathtt{T}_2^{\neq} \cup \mathtt{T}_{3,2}^{\equiv 1}$. First let us specify how we construct $\mathtt{T}_2^{=}$ and $\mathtt{T}_2^{\neq}$.

- We add a single copy of each (forced) edge in $\mathtt{T}_2^{\neq}$. Note that there are two parallel forced edges corresponding to every inequality constraint in $\mathtt{I}_2^{\neq}$; we add a single copy of both.
- For a variable set to 1, $\mathtt{T}_2^{=}$ contains exactly one copy of both unforced edges in $\mathtt{E}_2^{=}$ adjacent to the corresponding vertex in G.

Note that because every $\mathtt{I}_2^{=}$ and $\mathtt{I}_2^{\neq}$ clause is satisfied, for each $i \in [n]$ there is a single $b^* \in \{0,1\}$ such that $x_{i,j,b} = \mathbf{1}\{b = b^*\}$. So $\mathtt{T}_2^{=}$ contains exactly half the edges in $\mathtt{E}_2^{=}$. We can hence calculate $w(\mathtt{T}_2^{\neq}) = 20nk$, and $w(\mathtt{T}_2^{=}) = 11nk$. Next we describe how to construct $\mathtt{T}_{3,2}^{\equiv 1}$.

For each distinct clause of the form $\mathtt{P}_{3,2}^{\equiv 1}(x_{i_1,j_1,b_1}, x_{i_2,j_2,b_2}, x_{i_3,j_3,b_3})$, we consider the optimal spanning tour $Q^* = \arg\min_{Q \in \mathbb{Q}_{\mathbf{z}}} w(T)$, where $\mathbf{z} = (x_{i_1,j_1,b_1}, x_{i_2,j_2,b_2}, x_{i_3,j_3,b_3})$. We will add all *non-special* edges of Q^* on the corresponding contact vertices $\{x_{i_1,j_1,b_1}, x_{i_2,j_2,b_2}, x_{i_3,j_3,b_3}\}$, central vertex s, and auxiliary vertices $a_1, \ldots, a_{n_{\mathrm{aux}}}$ specific to the clause.

As an exception to the above, in the case that either of j_1, j_2, or j_3 is equal to $11k$, we will instead add two copies of *all* non-special edges, rather than just the edges in Q^*. The purpose of this step is to ensure connectivity of the tour; it has minimal impact on the cost of the tour, as k should be thought of as very large, and this step affects $o_k(1)$ fraction of the edges. We can bound the weight of edges added by

$$c(H) - (x_{i_1,j_1,b_1} + x_{i_2,j_2,b_2} + x_{i_3,j_3,b_3}) + 2w(H) \cdot \left(\mathbf{1}\{11k \in \{j_1, j_2, j_3\}\} + \mathbf{1}\{\mathtt{P}_{3,2}^{\equiv 1}(x_{i_1,j_1,b_1}, x_{i_2,j_2,b_2}, x_{i_3,j_3,b_3}) = 0\} \right),$$

where the second term accounts for the fact that we removed two special edges (each having weight $1/2$) for each variable that was set to 1.

Summing over all clauses in $\mathtt{I}_{3,2}^{\equiv 1}$, we get

$$w(\mathtt{T}_{3,2}^{\equiv 1}) \leq c(H) \cdot 2nk/3 - nk + 2w(H) \cdot (2n + \Delta),$$

where we used that the number of $\mathtt{I}_{3,2}^{\equiv 1}$ clauses is $2nk/3$, the fact that at most $2n$ clauses satisfy $11k \in \{j_1, j_2, j_3\}$, and at most Δ clauses are unsatisfied.

To complete the proof of Lemma D.2, we argue that T is indeed a valid spanning tour of G, which we do in the following two claims.

Claim D.1. *The number of edges of T adjacent to each vertex in G is even.*

Proof. The number of edges of $\mathtt{T}_2^{=}$ and $\mathtt{T}_2^{\neq}$ adjacent to each vertex in G is even by construction.

Every non-contact vertex is also adjacent to an even number of edges in $\mathtt{T}_{3,2}^{\equiv 1}$. Finally, since in every spanning tour in $\mathbb{Q}_{\mathbf{z}}$ for $\mathbf{z} \in \{0,1\}^3$, each special edge (which we abstained from adding to $\mathtt{T}_{3,2}^{\equiv 1}$) is used an even number of times, it also holds that every contact vertex is adjacent to an even number of edges in $\mathtt{T}_{3,2}^{\equiv 1}$. □

Claim D.2. *T has a single connected component consisting of all vertices in G.*

Proof. We will argue that each vertex is connected to the central vertex s.

- The contact vertices $x_{i,j,b}$ are connected to s using only edges in $\mathtt{T}_{3,2}^{\equiv 1}$ if either (1) $x_{i,j,b} = 0$, or (2) $j = 11k$.
- All checker vertices, and contact vertices set to 1 are connected to either $x_{i,11k,0}$ or $x_{i,11k,1}$ by edges in $\mathtt{T}_2^{=}$ and $\mathtt{T}_2^{\neq}$. Therefore they are also connected to s in T by the previous step.
- All auxiliary vertices in H are connected to all contact vertices as well as the central vertex 4 in any tour in $Q \in \mathcal{T}_{\mathbf{z}}$ for an assignment $\mathbf{z} \in \{0,1\}^3$. Consequently, they are also connected to either 4 or a contact vertex after removing any special edges from the Q. As a result, all auxiliary vertices in G are connected to either s, or to a contact vertex, which have all been previously shown to be connected to s using edges in T.

□

By construction, each forced edge is used at least once, and no edge is used more than twice. Therefore T is a valid spanning tour of G of weight at most $nk \cdot (30 + 2 \cdot c(H)/3) + 2(\Delta + 2n) \cdot w(H)$. □

Proof of Lemma D.3. Let us write $T = \mathtt{T}_2^{=} \cup \mathtt{T}_2^{\neq} \cup \mathtt{T}_{3,2}^{\equiv 1}$, where $\mathtt{T}_2^{=}$, $\mathtt{T}_2^{\neq}$, and $\mathtt{T}_{3,2}^{\equiv 1}$ are the edges in T belonging to $\mathtt{E}_2^{=}$, $\mathtt{E}_2^{\neq}$, and $\mathtt{E}_{3,2}^{\equiv 1}$ respectively. We define $\mathtt{U}_2^{=}$ to be $\mathtt{T}_2^{=}$ with all edges appearing twice removed.

We will assign $x_{i,j,b}$ based on the "local" view of the variable $x_{i,j,b}$ as a vertex in $\mathtt{U}_2^{=}$. In particular, let $l_{i,j,b}$ and $r_{i,j,b}$ denote the number of copies of the edges $(x_{i,j-1,b}, x_{i,j,b})$ and $(x_{i,j,b}, x_{i,j+1,b})$ in $\mathtt{T}_2^{=}$ respectively. We say that the variable $x_{i,j,b}$ is "honest" if $l_{i,j,b} = r_{i,j,b}$, and "dishonest" otherwise. We will always assign an honest variable $x_{i,j,b} = l_{i,j,b}$. Note that this is equivalent to the assignment $x_{i,j,b} = r_{i,j,b}$.

Next, we describe how to assign the dishonest variables. We will do this by specifying how to assign the dishonest variables appearing in each inequality and 3LIN(2) clause.

- For each inequality clause $\mathtt{P}_2^{\neq}(x_{i,j,0}, x_{i,M(j),1})$ in $\mathtt{I}_2^{\neq}$, note that the variables involved are either both honest, or both dishonest. In the case that both are dishonest, we assign each variable a random value in $\{0,1\}$, conditioned on $x_{i,j,0} \neq x_{i,M(j),1}$.

- For each 3LIN(2) clause $\mathtt{P}^{\equiv 1}_{3,2}(x_{i_1,j_1,b_1}, x_{i_2,j_2,b_2}, x_{i_3,j_3,b_3})$ in $\mathtt{I}^{\equiv 1}_{3,2}$ containing at least one dishonest variable, we arbitrarily set the dishonest variables involved, while ensuring that the clause is satisfied. Note that this is always possible as long as there is at least one such dishonest variable.

In order to analyze the number of unsatisfied clauses, we will need to define a few quantities.

- d_{ch} and d_{co} denote the number of dishonest checker and contact vertices respectively.
- p_{ch} denotes the number of inequality constraints in $\mathtt{I}^{\neq}_2$ such that both involved checkers are honest and assigned the same value.
- r is the number of unsatisfied clauses in $\mathtt{I}^{\equiv 1}_{3,2}$ where all involved variables are honest.

Claim D.3. *The expected number of unsatisfied clauses in $\mathtt{I}$ is at most $d_{ch} + d_{co} + 2p_{ch} + 2u_{ch} + 2r$.*

Proof. Let us list the collection of unsatisfied clauses. Using Definition D.5, we have the following:

- **Equality clauses:** By definition, the equality clause between two honest vertices is always satisfied. Every equality clause adjacent to a dishonest checker vertex is satisfied with probability exactly $1/2$ due to the random assignment of dishonest checkers.
  Out of the remaining edges, we claim that there is at most one unsatisfied edge adjacent to each dishonest contact vertex $x_{i,j,b}$. This is because if both $x_{i,j-1,b}$ and $x_{i,j+1,b}$ are honest, they must be assigned different values, as $r_{i,j-1,b} = l_{i,j,b}$ has parity opposite of $l_{i,j+1,b} = r_{i,j,b}$, since $x_{i,j,b}$ is dishonest. So, the expected number of violated equality clauses is at most $d_{ch} + d_{co}$.
- **Inequality clauses:** An inequality clause in $\mathtt{I}^{\neq}_2$ is unsatisfied when both checkers it involves are honest and assigned the same value. So, the number of violated inequality clauses is equal to p_{ch}.
- **3LIN(2) clauses:** By definition, all clauses in $\mathtt{I}^{\equiv 1}_{3,2}$ involving dishonest variables are satisfied. So the number of unsatisfied clauses in $\mathtt{I}^{\equiv 1}_{3,2}$ is equal to r.

Since the instance $\mathtt{I}$ is defined as $\mathtt{I}^{=}_2 + 2\mathtt{I}^{\neq}_2 + 2\mathtt{I}^{\equiv 1}_{3,2}$, the above bounds imply that the number of unsatisfied clauses in $\mathtt{I}$ is at most $d_{ch} + d_{co} + 2p_{ch} + 2r$. □

Claim D.4. $w(T) \geq nk \cdot (30 + 2 \cdot s(H)/3) + (d_{ch} + d_{co})/2 + p_{ch} + r$.

Proof. We will separately bound the weight of edges in $\mathtt{T}^{=}_2$, $\mathtt{T}^{\neq}_2$, and $\mathtt{T}^{\equiv 1}_{3,2}$. Let us denote by t_{co} the number of honest contact variables assigned 1. For $b \in \{0,1\}$ let p^b_{ch} denote the number of inequality clauses such that both involved checkers are assigned b. Note that $p^0_{ch} + p^1_{ch} = p_{ch}$. Finally, let u_{co} be the number of connected components in $\mathtt{U}^{=}_2 \cup \mathtt{T}^{\neq}_2 \cup \mathtt{T}^{\equiv 1}_{3,2}$ containing only edges from $\mathtt{E}^{\equiv 1}_{3,2}$.

- **Bounding $\mathtt{U}^{=}_2$:** By double-counting, we can write

$$w(\mathtt{U}^{=}_2) = \frac{1}{2}\sum_{i,j,b}\Big(l_{i,j,b} + r_{i,j,b}\Big) = \frac{1}{2}\left(\sum_{i,j:j|11,b}\Big(l_{i,j,b} + r_{i,j,b}\Big) + \sum_{i,j:j\nmid 11} v_{i,j}\right),$$

  where $v_{i,j} = \Big(l_{i,j,0} + r_{i,j,0} + l_{i,M(j),1} + r_{i,M(j),1}\Big)$. We will begin by bounding the first sum, that is, the sum over all contact variables. Observe that a dishonest contact variable must have $l_{i,j,b} + r_{i,j,b} \geq 1$, since $l_{i,j,b} \not\equiv r_{i,j,b} \pmod 2$. Similarly, a honest variable assigned 1 must have $l_{i,j,b} + r_{i,j,b} \geq 2$. So the first sum is at least $d_{co} + 2t_{co}$.
  Next, we will bound the second sum. We have that the sum of degrees of $x_{i,j,0}$ and $x_{i,M(j),1}$ in $\mathtt{U}^{=}_2$ is even; equivalently, $v_{i,j}$ must be even. So, $v_{i,j} \geq 2$ unless both involved checkers are assigned 0. Furthermore, we have $v_{i,j} \geq 4$ if both involved checkers are honest and assigned 1.
  Therefore

$$\sum_{i\in[n],j\in[11k]:j\nmid 11} v_{i,j} \geq 2 \cdot 10nk + 2 \cdot p^1_{ch} - 2 \cdot p^0_{ch}.$$

  Together, these bounds imply $w(\mathtt{U}^{=}_2) \geq 10nk + d_{co}/2 + t_{co} + p^1_{ch} - p^0_{ch}$.

- **Bounding $\mathtt{T}_2^{=} \setminus \mathtt{U}_2^{=}$:** Since $T = \mathtt{T}_2^{=} \cup \mathtt{T}_2^{\neq} \cup \mathtt{T}_{3,2}^{\equiv 1}$ is a spanning tour of G, we have that the total weight of edges in $\mathtt{T}_2^{=} \setminus \mathtt{U}_2^{=}$ is at least twice the number of connected components of $\mathtt{U}_2^{=} \cup \mathtt{T}_2^{\neq} \cup \mathtt{T}_{3,2}^{\equiv 1}$. So, $w(\mathtt{T}_2^{=} \setminus \mathtt{U}_2^{=}) \geq 2p_{ch}^0 + 2u_{co}$.
- **Inequality clauses:** Define $\kappa_{i,j}$ to be the total number of copies of both forced edges between $x_{i,j,0}$ and $x_{i,M(j),1}$. We can write
$$w(\mathtt{T}_2^{\neq}) = \sum_{i \in [n], j \in [11k] : j \nmid 11} \kappa_{i,j}.$$
Since T was a valid tour, we have $\kappa_{i,j} \geq 2$ for all such i, j. Furthermore, if either of the involved vertices is dishonest, the fact that the degree of that vertex in T is even implies that $\kappa_{i,j}$ is odd. As a consequence, we have $\kappa_{i,j} \geq 3$ for at least $d_{ch}/2$ such (i, j). This implies $w(\mathtt{T}_2^{\neq}) \geq 2 \cdot 10nk + d_{ch}/2$.
- **3LIN(2) clauses:** Let $\zeta = \mathtt{P}_{3,2}^{\equiv 1}(x_{i_1,j_1,b_1}, x_{i_2,j_2,b_2}, x_{i_3,j_3,b_3})$ be a clause in $\mathtt{I}_{3,2}^{\equiv 1}$. Let T_ζ be the set of edges in $\mathtt{T}_{3,2}^{\equiv 1}$ involved in clause ζ. We will extract a valid spanning tour Q of H from T_ζ, and then use the soundness property of H to argue that $w(T_\zeta)$ is large.
Formally, we will map the edges in T_ζ to a multiset of edges Q of the gadget. Note that Q is not necessarily a spanning tour of H yet. To resolve this, we add $l_{i_\ell,j_\ell,b_\ell} + r_{i_\ell,j_\ell,b_\ell}$ copies of the special edge $(\ell, 4)$ for each $\ell \in [3]$. For each remaining connected component that is not connected to the central vertex, we will add two copies of the special edge $(\ell, 4)$ for an arbitrary contact vertex ℓ in that connected component. Let us denote by u_{co}^{local} the number of such components.
Observe that the degree of all non-central vertices in Q is identical to the degree of the corresponding vertex in T. As a consequence, all non-central vertices have even degree in Q. Since the sum of degrees of vertices in a multigraph is always even, we automatically get that all vertices have even degree in Q. Furthermore, by definition Q is connected. Also note that Q uses each edge at most twice, and uses every forced edge at least once.
So, Q is indeed a valid spanning tour in H, implying $w(Q)$ is at least
$$s(H) + k_1^H(Q)/2 + \mathbf{1}\{k_1^H(Q) = 0\} \cdot \mathbf{1}\{k_2^H(Q) \text{ is even}\}.$$
Subtracting the contribution of the special edges we added to Q, we get
$$\begin{aligned} w(T_\zeta) &= w(Q) - k_1^H(Q)/2 - k_2^H(Q) \\ &\geq s(H) - u_{co}^{\text{local}} - (k_2^H(Q) - u_{co}^{\text{local}}) + \mathbf{1}\{k_1^H(Q) = 0\} \cdot \mathbf{1}\{k_2^H(Q) \text{ is even}\}. \end{aligned}$$
Note that if $u_{co}^{\text{local}} = 0$, the indicator $\mathbf{1}\{k_1^H(Q) = 0\} \cdot \mathbf{1}\{k_2^H(Q) \text{ is even}\}$ exactly indicates whether $\zeta = 1$, that is, whether the clause is satisfied. Therefore, the above is at least
$$s(H) - 2u_{co}^{\text{local}} - (k_2^H(Q) - u_{co}^{\text{local}}) + \zeta$$
Summing over all $2nk/3$ clauses c in $\mathtt{I}_{3,2}^{\equiv 1}$, we get
$$w(\mathtt{T}_{3,2}^{\equiv 1}) \geq (2nk/3) \cdot s(H) - 2u_{co} - t_{co} + r,$$
where we used that the sum of u_{co}^{local} equals u_{co}, along with the fact that the sum of $k_2^H(Q) - u_{co}^{\text{local}}$ is the total number of contact vertices $x_{i,j,b}$ satisfying $l_{i,j,b} = r_{i,j,b} = 1$, which equals t_{co}.

Together, these imply the lower bound $w(T) = w(\mathtt{U}_2^{=}) + w(\mathtt{T}_2^{=}) + w(\mathtt{T}_2^{\neq}) + w(\mathtt{T}_{3,2}^{\equiv 1}) \geq nk \cdot (30 + 2 \cdot s(H)/3) + (d_{ch} + d_{co})/2 + p_{ch} + r$. □

Since T has weight at most $nk \cdot (30 + 2 \cdot s(H)/3) + \Delta$, Claim D.4 implies that $(d_{ch} + d_{co})/2 + p_{ch} + r \leq \Delta$. Claim D.3 directly shows that at most $d_{ch} + d_{co} + 2p_{ch} + 2r \leq 2\Delta$ clauses are unsatisfied in expectation over the randomness in the assignment $\mathbf{x}$; this proves that there exists an assignment leaving at most 2Δ clauses unsatisfied. □

D.2 Description of our gadget

In this section, we provide the equation gadget (Definition D.3 and Figure 7a) found by AlphaEvolve, and instantiate Theorem D.2 with this gadget to obtain Theorem 4.1. The gadget $H = ([3+1+n_{\text{aux}}], E_u \cup E_f, w)$ will have $n_{\text{aux}} = 4$ auxiliary vertices. As in Appendix C.1, we will specify the edges in the gadget by a weighted edge list, where an edge is represented as a tuple $(a, b, w(a, b))$, representing an edge between a and b of weight $w(a, b)$. The unforced edges E_u contain the special edges (see Definition D.3), along with the following non-special unforced edges.

```
[(5, 6, 1.0), (6, 7, 1.0), (7, 8, 1.0), (8, 5, 1.0)]
```

The forced edges E_f are defined as follows.

```
[(1, 5, 1.0), (1, 5, 1.0), (2, 6, 1.0), (2, 6, 1.0), (3, 7, 1.0), (3, 8, 1.0)]
```

Below we calculate the various parameters associated with H.

Lemma D.4. *Let H be the equation gadget defined above. We have $c(H) = s(H) = 10$, and $\max_{e \in E_u \cup E_f} w(e) = 2$.*

We note that this is in alignment with the discussion in Section 4; every satisfying assignment to $\mathrm{P}_{3,2}^{\equiv 1}$ admits a tour of cost at most $c(H) = 10$, whereas for every tour Q corresponding to an unsatisfying assignment, we have $k_1^H(Q) = 0$ and $k_2^H(Q)$ is even; hence Q has weight at least $s(H) + \mathbf{1}\{k_1^H(Q) = 0\} \cdot \mathbf{1}\{k_2^H(Q) \text{ is even}\} = 11$. We can now prove Theorem 4.1.

Proof of Theorem 4.1. We apply Theorem D.2 with the above gadget, and use Lemma D.4 to conclude that it is NP-hard to approximate MWST_f within

$$\frac{91 + 2 \cdot s(H)}{90 + 2 \cdot c(H)} - \varepsilon' = \frac{111}{110} - \varepsilon'$$

for any $\varepsilon' > 0$. Using Lemma D.1 along with the fact that MWST is equivalent to metric TSP, we conclude that it is also NP-hard to approximate metric TSP within $111/110 - \varepsilon$ for any $\varepsilon > 0$. □

D.3 Equation gadget for predicate $x + y + z \equiv 0 \pmod 2$

In this part, we provide a detailed comparison of our gadgets with those in [CC20]. As mentioned in Section 4, [CC20] operated with a version of Hybrid-3LIN(2) containing of $\mathrm{P}_{3,2}^{\equiv 0}$ clauses (defined as $\mathrm{P}_{3,2}^{\equiv 0}(x, y, z) = \mathbf{0}\{x + y + z \equiv 0 \pmod 2\}$) rather than $\mathrm{P}_{3,2}^{\equiv 1}$. As a CSP, this is completely equivalent to our Hybrid-3LIN(2) instances containing $\mathrm{P}_{3,2}^{\equiv 1}$ clauses applied to the pointwise negation of the variables. However, as the $\{0, 1\}$-assignment to a variable is encoded in a way that is not symmetric under negation (see Definitions D.3 and D.4), one needs a formally different notion of soundness and completeness in their setting. Since this notion is not critical to the correctness of our main results, we informally describe them. For an equation gadget H, the modified completeness $c'(H)$ is defined by maximizing over all $z \in (\mathrm{P}_{3,2}^{\equiv 0})^{-1}(1)$ rather than $C = (\mathrm{P}_{3,2}^{\equiv 1})^{-1}(1)$ as in Definition D.4. The modified soundness $s'(H)$ is defined by replacing the $\mathbf{1}\{k_2^H(Q) \text{ is even}\}$ term in the definition of $s(H)$ by $\mathbf{1}\{k_2^H(Q) \text{ is odd}\}$.

Similarly to Theorem D.2, one can show that it is NP-hard to approximate MWST_f within $(91 + 2 \cdot s'(H))/(90 + 2 \cdot c'(H)) - \varepsilon$ for any equation gadget H. [CC20] provide an equation gadget H (see Figure 8a) achieving $s'(H) = c'(H) = 13$, leading to their inapproximability of $117/116 - \varepsilon$ for metric TSP.

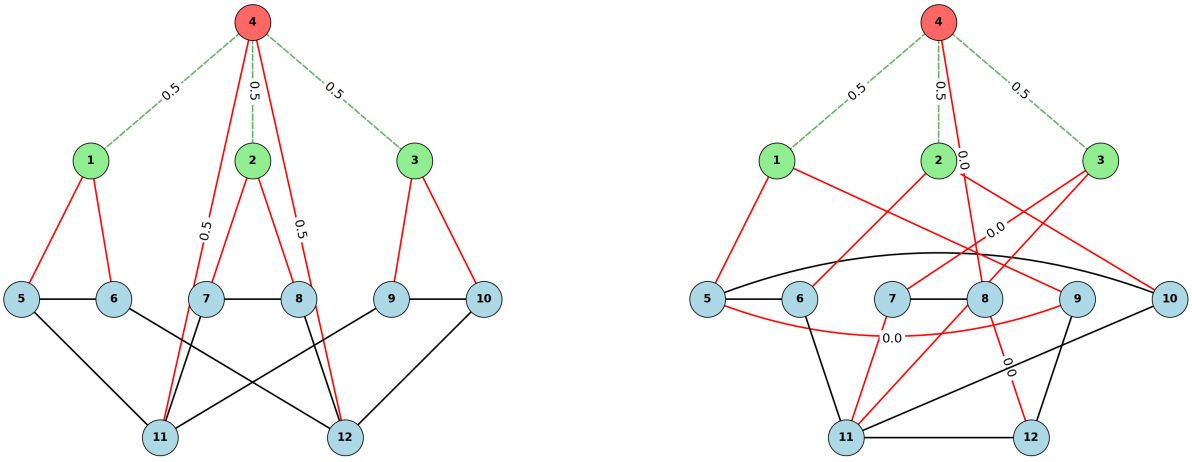
As a final note, we used AlphaEvolve to find an equation gadget $H = ([3 + 1 + n_{\text{aux}}], E_u \cup E_f, w)$ (see Figure 8b) for $\mathrm{P}_{3,2}^{\equiv 0}$ clauses on $n_{\text{aux}} = 8$ auxiliary vertices. Below we describe H in the same format as Appendix D.3. The unforced edges in E_u are the following.

```
[(5, 6, 1.0), (7, 8, 1.0), (10, 11, 1.0), (11, 12, 1.0), (12, 9, 1.0), (5, 10,
    1.0), (6, 11, 1.0)]
```

The forced edges E_f are defined as follows.

```
[(1, 5, 1.0), (2, 6, 1.0), (3, 7, 0.0), (4, 8, 0.0), (5, 9, 0.0), (7, 11, 1.0)
   , (8, 12, 0.0), (9, 1, 1.0), (10, 2, 1.0), (11, 3, 1.0)]
```

This gadget achieves $s'(H) = c'(H) = 10$, giving an alternative approach to prove our main result Theorem 4.1. However, this gadget is significantly more complicated than the $\mathtt{P}_{3,2}^{\equiv 1}$-based equation gadget from Lemma D.4 achieving $s(H) = c(H) = 10$, which is why we chose to write our proofs in terms of $s(H)$ and $c(H)$.



(a) The equation gadget from [CC20].

(b) Equation gadget found by AlphaEvolve.

Figure 8: Equation gadgets in the MWST_f instance. Vertices $\{1, 2, 3\}$ represent variables in the 3LIN(2) equation $\mathtt{P}_{3,2}^{\equiv 0}$. The red edges edges represent the forced edges. The dashed green edges represent the special edges. All edges without an explicitly labeled weight have weight 1.